UPMC UFR 919

M2

ABDR

Administration des bases de données réparties

Hubert Naacke

Site :
http://www-master.ufr-info-p6.jussieu.fr/2006/Ext/naacke/**abdr2008**

Sommaire :

ABDR : Intro et Outils

Hubert Naacke

Laboratoire d'Informatique de Paris 6

Objectifs

- Comprendre le fonctionnement d'un SGBD
- Comprendre les problèmes de performance
  - Diagnostic
  - Résolution
- Comprendre les fonctionnalités/limitations
  - Traitement de requêtes réparties
- Acquérir un sens pratique
  - capacité de réaction, quel que soit le SGBD

Planning

- Cours
  - Intro: Etude de cas, benchmark TPC
  - Stockage
  - Indexation
  - Contrôle de concurrence, transactions
  - Requêtes
- Pratique
  - Oracle Database: version 10G
  - En option
    - IBM DB2 Universal Database
    - Apache Derby, MonetDB

Evaluation

- Projet
  - 30h perso
- Examen
  - 2h, mi janvier
- CC
  - TME
  - Participation

Projet

- Banc d’essai
  - Générateur de BD
  - Générateur de charge
  - Surveillance, diagnostic

Bibliographie

- P. Bonnet, D. Shasha
  - Livre : Database Tuning
- P. Rigaux (Univ. Paris Dauphine)
  - cours en ligne
- Documentation Oracle10G
  - otn.oracle.com

Pratique : TME avec le SGBD Oracle 10G

Oracle

- SGBD relationnel
- Outils de développement
  - IDE : jdeveloper
  - Modules: packages PL/SQL, lib java
  - Dvpt de sites web : Portal
- Outils d'administration
  - Architecture 2 tiers
    - DBA Studio (java) + SGBD oracle9i
  - Architecture 3 tiers : Enterprise Manager
    - Appli web : navigateur + apache + oracle10g

Installer Oracle10G

- Logiciel sur CD ou otn.oracle.com
  - Logiciel Oracle 10G + tous les livres de la doc
  - Complément facultatif: Companion, Jdeveloper 10G
- Pré-requis
  - OS, utilisateur
- Installation
  - Le logiciel serveur
  - Connectivité client/serveur
  - Les instances (une instance = 1 BD)
- Post installation
  - Démarrage automatique
  - Archivage
  - Sécurité

Documentation Oracle

- Documentation Library
  - Retrouver rapidement le livre utile
    - 1 onglet par thème
  - Book List (onglet *books*)
    - raccourcis
  - Recherche par mot-clé
  - Dans chaque livre : content, index
- Manuel d'Installation (Installation Doc)
  - Spécifique pour chaque OS (linux64, linux32, …)
  - Résumé pour une installation rapide
    - Oracle Database Quick Installation Guide

Outils d'administration des BD réparties

Les principaux outils

- Accès au SGBD
  - Admin classique: Enterprise Manager, interface web
    - emctl [status|start|stop] dbconsole
  - Requêtes SQL, programme en langage PL/SQL
    - Interface web: **i**sqlplus IHM ergonomique
    - Interface textuelle, ligne de commande: sqlplus
  - Impor/Export de données : sqlloader
- Besoin d'un accès plus avancé
  - Accès à distance
  - Accès à plusieurs SGBD
  - Automatiser l'accès
    - Traitements automatiques, fréquents
    - Traitements personnalisés

Accès C/S

- Un SGBD est un logiciel C/S
  - Protocole non standard (ex: Net8 pour Oracle)
  - Accès multi utilisateurs
  - Accès local ou à distance
  - Indépendant de l'OS et du langage d'implem
    - Client Java sur Mac / SGBD C sur Win
    - Client C sur Win / SGBD C sur linux
    - Client Java sur machine M1 / SGBD C sur M1

Protocole C/S : Net8

- Protocole client/serveur applicatif
  - Transparent Network Substrate (TNS)
  - Repose sur un protocole de communication standard
    - Accès à distance TCP, …
    - Accès local : communication inter processus, utilise la mémoire partagée
- Connexion en deux étapes
  1 - Ouvrir une session
  2 - Envoyer des traitements aux SGBD
- Module d'écoute (listener)
  - Gère l'étape 1 : un client demande l'ouverture d'une session
  - Initie l'étape 2: le module d'écoute met en contact le SGBD avec le client
- Centralisateur (connection manager)
  - Gère entièrement les étapes 1 et 2 : intermédiaire permanent entre le client et le SGBD
  - Avantage: contrôle d'accès
  - Inconvénient: goulot

Serveur Net8: Module d’écoute

- Application
  - Située sur la même machine que le SGBD, mais processus indépendant
- Combien de modules d’écoute ?
  - Un module d’écoute par machine :
    - Le module d’écoute redirige les demandes vers tous les SGBD de la machine
  - Plusieurs modules d’écoute
    - Un module par SGBD
    - Plusieurs modules d’écoute pour un SGBD
- Contrôler l’accès à distance
  - Pas d’accès distant sans module d’écoute
  - Accès local possible sans module d’écoute
    - combiner l’accès local avec un accès distant par un protocole autre que net8 (ex. ssh)
- Configuration
  - Fichier $ORACLE_HOME/network/admin/listener.ora
  - Protocole de comm (ex: TCP), n° du port d’écoute : 152 1 par défaut
  - Taille de la file d’attente, liste des instances utilisant ce module d’écoute

Module d’écoute: usage

- Contrôle local
  - **lis**ten**e**r **c**on**t**ro**l** : **lsnrctl** [status stop start]
    - lsnrctl status : indique si le module d’écoute est actif
    - lsnrctl start : démarre et indique les instances servies
  - Outil de contôle local fourni par l’OS
    - Win : net start *nom de service du module d’écoute*
      puis netstat –ap tcp (Win)
    - Linux : netstat –lt (Linux)
- Contrôle à distance
  - tnsping *hôte* :
    - indique si le module d’écoute est actif sur une machine
    - *hôte* : nom FQN, adresse IP, nom win
  - telnet
- Inscription: Associer une instance à un module d'écoute
  - Process Monitor: PMON, vérification périodique (1/minute)
  - alter system register

SGBD: serveur dédié/partagé

- Un programme client communique avec
  - Un processus SGBD (Linux)
  - Un processus léger, thread SGBD (Win)
- Serveur dédié
  - Un processus pour chaque client
- Serveur partagé
  - N clients par processus
  - Utilise un dispatcher
  - Multiplexage: n sessions Net8 dans 1 session TCP
  - Pooling : réduire la latence à l'ouverture d'une session

Client Net8

- API pour le client
  - Langage C: OCI
    - Win : pilote ODBC
  - Java: pilote JDBC
- Usage
  - Installation immédiate (instant client, depuis 10G)
  - C : indiquer les librairies (LD_LIBRARY_PATH)
  - Java
    - Application
      - indiquer l'archive du pilote
        - CLASSPATH=ocjdbc14.jar (JDK > 1.4, Oracle >8i)
        - CLASSPATH=classes12.jar (JDK<1.4, Oracle > 8i)
        - CLASSPATH=classes12.zip (JDK<1.4, Oracle < 9i)
    - Applet : le pilote peut être téléchargé juste avant l'utilisation

Net8: Interface OCI

- OCI : Oracle Call Interface est l'implem cliente de Net8
- OCI utilisé par sqlplus et pour les requêtes et trans réparties
  - Ex: sqlplus *utilisateur/mot_de_passe@nom_service_réseau*
- Paramètres
  - Fichier tnsnames.ora dans le répertoire référencé par TNS_ADMIN

```
nom_de_service_réseau =
  (description =
     (address_list =
        (address = (protocol = tcp)(host = machine)(port = 1521)))
     (connect_data= (SID = base01) ) )
```

  - Identifier un SGBD sans ambiguïté
    - Remplacer sid par service_name (fait référence au nom global de l'instance)
  - Définir si un processus dédié est alloué au client
    - Ajouter server = dedicated ou shared
  - Fichier sqlnet.ora : nom de domaine par défaut

Net8: Interface ODBC

- ODBC : couche au dessus de OCI
- Paramètres
  - Panneau de configuration
    - (Outil d'administration)
      - Source ODBC
        - Pilote ODBC d'Oracle, définir le nom ODBC du SGBD
- Accès depuis le langage C
- Accès possible depuis java
  - Pilote = passerelle JDBC/ODBC
    - L'implem de Driver est *sun.jdbc.odbc.JdbcOdbcDriver*
    - Syntaxe de l'URL de connexion : jdbc:odbc:*nom*
      - *nom* = nom ODBC du SGBD

Net8: Interface Java

- Pilote = implem du client Net8
  - L'implem du Driver est jdbc.driver.OracleDriver
- Client
  - 100% Java
    - URL: jdbc:oracle:thin:@*hôte*:*port*:*SID*
    - Ex: jdbc:oracle:thin:@pc8:1521:base08
  - Passerelle JDBC/OCI
    - URL : jdbc:oracle:**oci8**:*nom_de_service_réseau*
    - Utilise le fichier tnsnames.ora

Accès C/S : cas d'utilisation

- Client sqlplus sur linux
  - installation
    - Décompresser instant-client dans un répertoire
    - Ajouter le répertoire à la liste LD_LIBRARY_PATH
    - Créer un fichier tnsnames.ora
    - Positionner TNS_ADMIN
  - Usage
    - sqlplus system/mot_de_passe@nom_service_réseau
- Client Java sur linux
  - Installation
    - JDK (java.sun.com), copier le pilote dans un répertoire
    - export CLASSPATH**=.:***répertoire_du_pilote*/ojdbc14.jar
  - Usage
    - java Application

Accès C/S : transfert de port

- Transférer une appli web (protocole http)
  - ssh avec tunnel pour le transfert local
    - accès à EM : http://*client*:5500
    - Isqlplus : http://*client*:5560
- Transférer le module d'écoute
  - ssh : pb le tunnel 1521 ne suffit pas
- Transférer cman : ok
- Transférer l'affichage
  - ssh avec tunnel pour transférer le serveur graphique
    - X11 ou VNC
  - bureau distant (WinXP)

23

Application répartie

- Plusieurs applications clientes
- Appli cliente + appli dans le SGDB
  - Procédure stockée
    - PL/SQL, Transat SQL, standard Persistent Stored Module
    - Java : pilote JDBC «interne» au SGBD
      - Communication rapide, sans Net8, entre
        - La JVM
        - L'interpréteur PL/SQL
        - Le moteur de requêtes
- Appli multi-utilisateurs
  - Java : une sesssion = un thread
  - PL/SQL : une session = 1 job
  - sqlplus : une session = une fenêtre de terminal

24

PL/SQL

- Lire
  - Manuel
    - PL/SQL User's Guide and Reference
  - API
    - Supplied PL/SQL Packages and Types Reference
    - PL/SQL Packages and Types Reference
      - + de 190 packages
      - Exple: dbms_utility.get_time()

ABDR: Projet

Projet: Objectif

- Concevoir et réaliser un démonstrateur
- Démontrer une fonctionnalité ‘avancée’ du SGBD qui accélère le traitement des transactions
- Avantages, inconvénients
- Insister sur l’amélioration par rapport à l’existant

Implication: 30h perso minimum

Projet: Modules à concevoir

- Banc de test
  - Générateur de base de données
  - Générateur de charge
  - Surveillance, diagnostic

Générateur

- Générateur de BD
  - Schéma table, contraintes
  - Méthode d'accès : index
  - Données
    - Valeur des attributs
    - Distribution aléatoire : uniforme, non-uniforme, zipf, …
    - Attributs non corrélés
- Générateur de charge
  - émule les applications clientes
    - Nombre d'applications simultanées (terminaux)
    - Requêtes paramétrées
    - Transactions, niveau d'isolation
    - Attente

Surveillance, diagnostic

- Etat du SGBD
  - État instantané du SGBD : vues V$...
  - Etat à postériori : trace, log, plan d'exécution
- Etat des appli clientes et du réseau
- Mesure
  - Unité de mesure: nb de pages, nb d'octets, nb de verrous
  - Temps de réponse
    - Client, comm C/S, SGBD
    - Temps d'attente : cause ?
  - Débit : nb de transactions par minute (tpm)
    - Stabilité: débit indépendant de la durée de l'expérience

Démarche

- Progression par étape. 1 étape =
  - Scénario de référence
    - Constater la performance initiale
    - Proposer une solution pour améliorer la perf
  - Scénario de validation expérimentale
    - Mesurer l'amélioration
    - Conclure
- Terminer une étape avant d'aborder la suivante

Rapport de projet

- Plan
  - Intro
  - Description générale
    - Schéma de la base
    - Traitements (requêtes, transactions)
    - Mesure de performance
  - Fonctionnalité 1
    - Mise en évidence
    - Manque
    - Amélioration
  - ...
  - Fonctionnalité n
  - Conclusion
- Forme: rédaction, clarté

7

Sujet : Projet 2008

- Contexte : le benchmark TPC-C
  - Cf. www.tpc.org rubrique TPC-C
- Impact de la répartition
  - Comparer BD centralisée / BD répartie sur 2 à 4 machines
  - Réaliser le gérant de la répartition
- Impact du stockage
  - Proposer une solution pour réduire l'usage du disque
- Impact du contrôle de concurrence
  - Proposer une solution pour réduire le surcoût lié au contrôle de concurrence
- Disponibilité
  - Proposer une solution pour surmonter la panne d'une base

8

Bases de données réparties pour le commerce en ligne

Etude de cas : l'architecture de Ebay

Charge applicative

- Clientèle : 248 millions d'utilisateurs
- Tailles des données
  2 000 tera octets
    - 200 fois la taille de la librairie du Congrès
  Photos : + d'1 milliard
  100 millions d'articles en vente dans 50 000 catégories
- Requêtes
  Demandes : 1 milliard de pages web par jour
  ➔ 44 milliard de requêtes SQL par jour
  dont 130 millions d'appels de procédure par jour
- Transactions commerciales
  Montant des négociations : 1800 $ par seconde
- Application en constante évolution
  + de 300 nouvelles fonctionnalités par trimestre
  Ajout de 200 000 lignes de code par mois

Besoins

- Passage à l'échelle (scalabilité)
  - L'utilisation des ressources doit augmenter proportionnellement avec la charge
  - Croissance forte : prévoir une multiplication par 10 des volumes : données, utilisateurs, communication
- Disponibilité (24 x 7 x 365)
  - Tolérance aux pannes
  - Fonctionnement en mode dégradé
  - Reprise après panne
  - 28h hors service en 15 ans → 99,9978 % de dispo
- Latence
  - temps de réponse constaté par les utilisateurs
  - temps de transfert des données entre 2 points de l'infrastructure
- Gestion de l'infrastructure
  - Simplicité
  - Maintenabilité
  - Inspection : diagnostic rapide
- Coût
  - Complexité et effort de développement
  - Coût opérationnel

Stratégies

Concevoir une solution guidée par les principes suivants :

1. Utiliser la fragmentation
2. Communiquer en mode asynchrone
3. Automatiser les tâches
4. Prévoir les pannes

Stratégie 1

Utiliser la fragmentation

Fragmentation

- Diviser chaque problème en sous problèmes
  - S'appuyer sur les solutions existant pour des problèmes de petite dimension : charge, donnée, cas d'usage
- Raisons
  - Passage à l'échelle :
    - Multiples ressources indépendantes
  - Disponibilité : pannes isolées
  - Evolutivité : découpler les fonctionnalités
    - Application modulaire
  - Coût : matériel banalisé → faible coût unitaire
- Type de fragmentation
  - Fragmentation fonctionnelle
  - Fragmentation horizontale

Fragmentation des données (1)

- Fragmentation fonctionnelle
  - Un groupe de bases de données par domaine
  - BD1: Utilisateurs BD2 : Articles ...
  - BD3: Transactions... BD4 : ... ...
- Regrouper les données à partir du modèle conceptuel
  - Cardinalité, Associations, modèle d'usage
- Hôtes logiques
  - Indépendance entre l'organisation physique et la représentation logique des données
  - Permet de réorganiser les BD sans modifier le code des applications

→ 400 machines hébergeant 1000 bases de données

Fragmentation des données (2)

- Fragmentation horizontales selon la clé d'accès la plus utilisée
  - Hachage (modulo) sur la clé primaire
    - n°d'articles, identifiant d'utilisateur
  - Valeur d'un attribut
    - catégorie
  - Par intervalle
    - Prix, date
- Surcouche de traitement réparti du SQL
    - Rend la répartition transparente aux applications
    - Achemine les instructions SQL vers le fragment adéquat
    - Configurable pour équilibrer la charge

Traitement de transactions pour les données fragmentées

- Peu de transactions ACID
- Stratégie transactionnelle
  - Eviter les transactions applicatives et la validation répartie
  - Une seule opération SQL par transaction (fréquent)
    - Modifie un seul n-uplet sur une seule base
  - Quelques transactions plus complexes (rare)
    - Modifie plusieurs n-uplets sur plusieurs tables d'une seule base
    - Mise en oeuvre par une procédure traitée par le SGBD (bloc PL/SQL anonyme)
- Cohérence garantie malgré l'absence de transaction
  - Ordonnancement judicieux des opérations
  - Restauration à l'aide d'une couche de communication fiable
    - Événements de restauration asynchrones
    - Procédure de réconciliation
    - Diffusion atomique de séquence d'événements
- Avantages
  - Pas d'interblocage
  - Supprime le besoin de disponibilité simultanée des participants d'une transaction
  - Maximise l'exécution simultanée des écritures
- Différents niveaux de cohérence requis selon les transactions

Fragmentation de l'application (1)

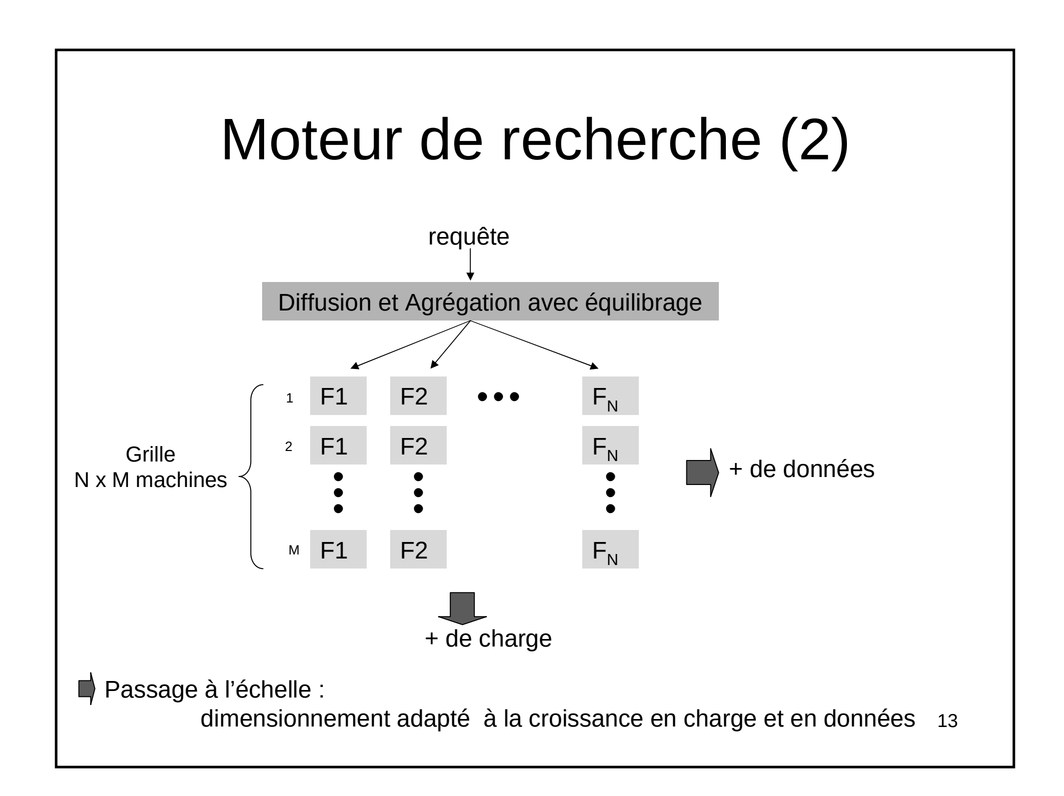
- Fragmentation fonctionnelle
  - Séparer les fonctionnalités dans des groupes applicatifs distincts
    220 groupes pour 16 000 serveurs d'application
    G1 : Vente G2 : Recherche ...
    G3 : Voir les articles G4 : Enchères ...
  - Paralléliser le développement, le déploiement et le contrôle des modules applicatifs
  - Minimise les dépendances entre les BD et les ressources
    - infrastructure BD dédiée pour un type d'application
      - BD en lecture seule pour la recherche et la consultation d'articles
      - BD en lecture/écriture pour les enchères et les ventes
- Fragmentation horizontale
  - Un groupe est constitué de N serveurs d'applications identiques
  - Applications sans état
    - clonage sans partage de données entre applications

Fragmentation de l'application (2)

- Couche applicative sans état
  - L'accès partagé à des objets applicatifs ne passe pas à l'échelle
  - Aucune donnée n'est transmise entre deux groupes applicatifs
- La session d'un utilisateur transite par plusieurs groupes applicatifs
  - Recherche puis visualisation d'un article puis enchères, puis paiement.
- Conserver l'état d'une session
  - URL : petite taille
  - Cookies : taille moyenne
  - BD en mémoire : grande taille
    - historique de navigation à travers plusieurs pages de formulaires

Moteur de Recherche (1)

- Fragmentation fonctionnelle
  - BD spécifique pour la recherche : index séparé des données
  - Accès en lecture seule séparé des transactions de lecture/écriture
  - Un moteur de recherche par domaine (utilisateur, article, ...)
- Fragmentation horizontale de l'index
  - Index divisé en N fragments
    - Taille maxi d'un fragment pouvant tenir en mémoire
  - Chaque fragment est répliqué M fois
- Déploiement
  - Sur une grille de N colonnes x M machines
- Accès
  - Chaque requête est diffusée vers les N fragments
  - Pour chaque fragment : choisir la réplique la moins chargée
  - Agrégation des N résultats partiels



Moteur de recherche (3)

- Index adapté aux critères des requêtes
  90% des recherches sur moins de 2 mots clés
  Tri par prix ou date
- Requête tronquée
  - L'utilisateur lit seulement les 10 premiers éléments du résultat
    - Exemple : les 10 premiers articles dont le titre contient le mot 'M', triés par date d'expiration
  - Tri local sur chaque fragment
    - Chaque fragment envoie seulement 10 nuplets
  - Agrégateur
    - Fusion de N listes de 10 éléments triées

14

Stratégie 2

Communiquer en mode asynchrone

Asynchronisme : avantages (1)

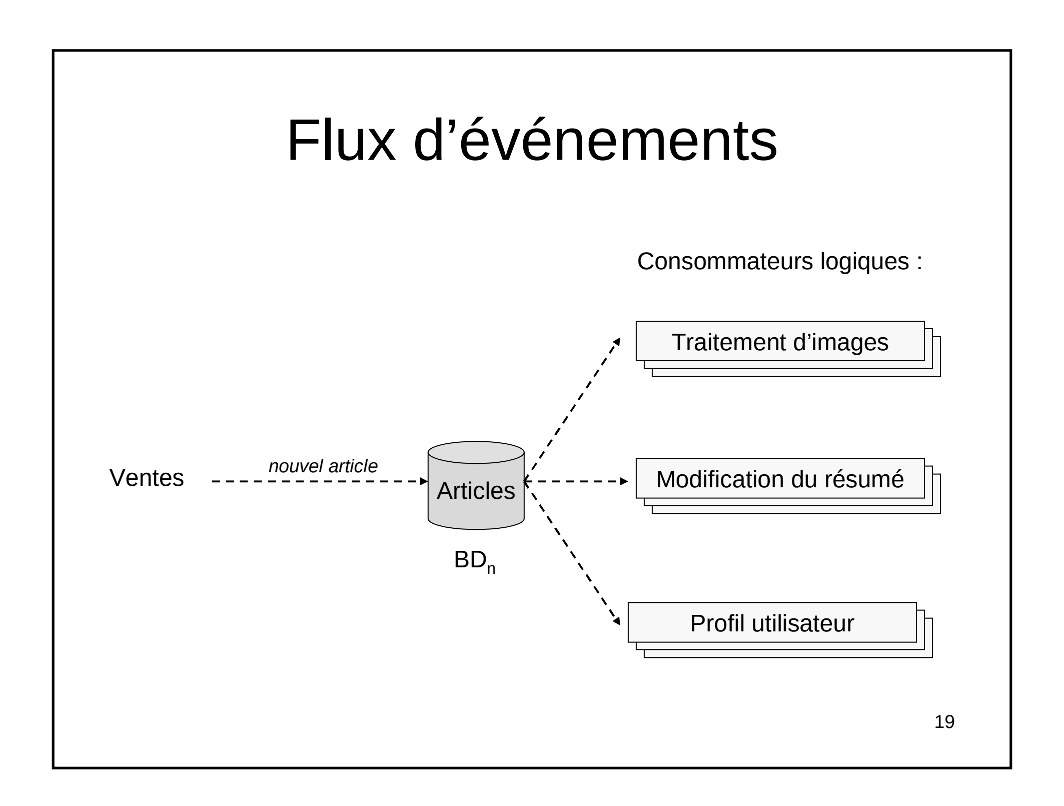
- Traitement asynchrone (sans attente bloquée)
  - Généraliser le choix d'une communication asynchrone autant que possible
  - Communication asynchrone pour l'intégration des divers éléments de l'architecture
- Avantages
  - Passage à l'échelle : dimensionnement indépendant de chaque élément
    - BD, groupe d'application
  - Disponibilité
    - Plusieurs états de disponibilité peuvent cohabiter
    - Caractéristiques de disponibilité dédiées pour une application
      - Haute disponibilité seulement pour les éléments qui l'exigent
      - Best effort pour les autres éléments
  - Invocation tolérant l'indisponibilité
    - plusieurs tentatives pendant un certain délai

Asynchronisme : avantages (2)

- Latence
  - Retarder l'exécution de certains traitements pour améliorer le temps de réponse observé par l'utilisateur
  - Offre la possibilité d'effectuer des traitements longs en arrière plan
    - durée > max toléré par l'utilisateur
- Coût opérationnel
  - Les pics de charge sont 'lissés' dans le temps
    - Les files de messages asynchrones servent de tampons pour absorber les pics de charge
    - Un pic de charge ne provoque pas la saturation du système.
    - Infrastructure moins chère
      - Dimensionnée pour supporter une charge moyenne et non une charge maximale

Flux d'événements
Diffusion de messages

- Une transaction génère un événement
  + de 300 types d'événements
    - Ajout d'un article, nouvelle enchère, article vendu, ...
  - Événement persistant
    - création incluse dans la transaction (bloc PL/SQL)
- Un consommateur s'abonne aux événements
  - Plusieurs consommateurs logiques pour un type d'événement
  - Diffusion fiable des événements
    - au moins une fois, sans garantir l'ordre d'arrivée
- Traitement des événements sans ordre ni unicité
  - Traitement idempotent
    - app(evt) = app(app(evt))
  - Evénement = signal
    - ne contient pas les données mises à jour.
    - Accès supplémentaire à la BD pour connaître les modifications effectuées

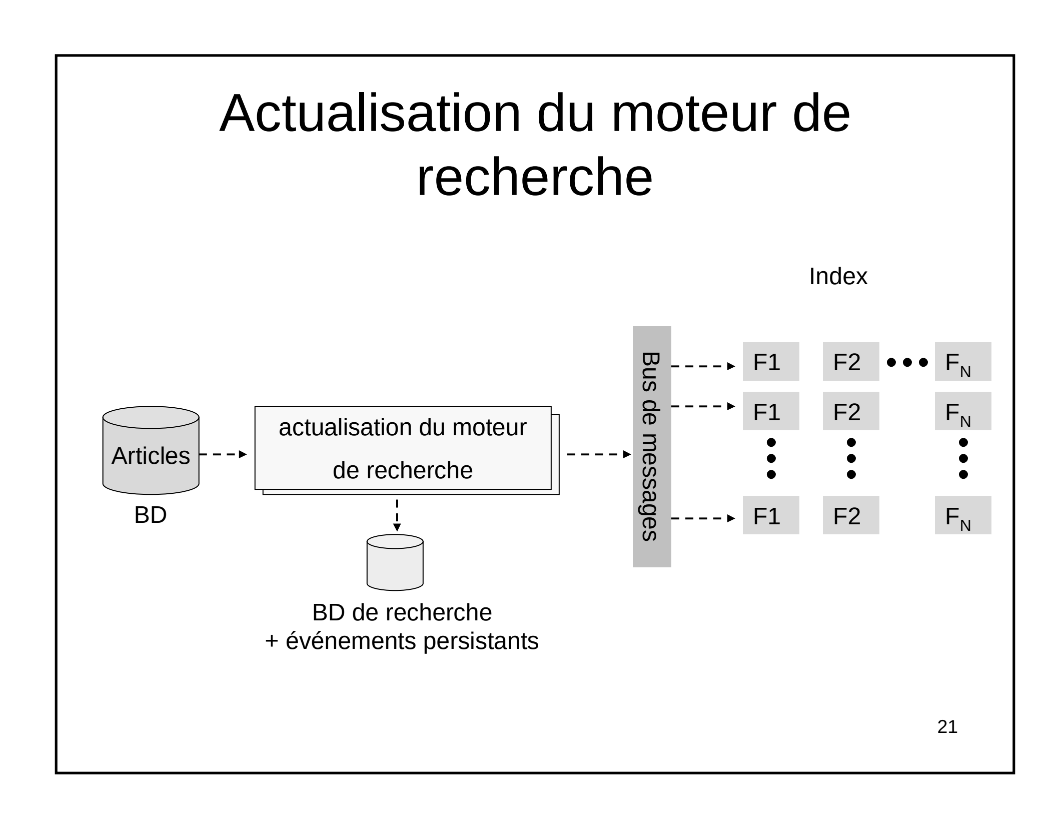


Mise à jour de l'index du moteur de recherche

- Evénement
  - Généré par une écriture dans la BD
    - Exple: transaction d'ajout d'un article
- Traitement d'un événement
  - Transformation des données
    - Normalisation du texte, ajout de métadonnées
  - Diffusion fiable
    - Publication vers tous les fragments de l'index sur la grille
    - Stockage persistant dans une BD auxiliaire
    - Renouveler les envois n'ayant pas abouti
- Chaque fragment de l'index écoute l'arrivée des messages
  - Filtre : ignore les messages ne concernant pas le fragment
  - Mise à jour de l'index en mémoire
  - Demande le renvoi des messages perdus

20

Actualisation du moteur de recherche



Traitement des lots

- Lots (batch)
  - planifiés et traités sur des ressources hors ligne
  - Période : jour, semaine, mois
  - Manipulation intensive des données
    - Parcours séquentiel d’une table entière
  - Exemples
    - Calculer des profils et des recommandations
      - Lire tous les articles, catégories, recherche, ...
    - Importer des données externes
      - Catalogue, ...
    - Calcul décisionnels
      - Top 10 des meilleures ventes par catégorie
    - Archivage / suppression des articles vendus
  - Génère d’autres traitements de fond via des événements

Stratégie 3

Automatiser les tâches

Automatisation

- Préférer les systèmes automatiques et adaptatifs aux systèmes manuels
- Avantages
  - Passage à l'échelle
    - Ajout de ressources matérielles sans ajouter d'administrateur
  - Disponibilité et Latence
    - Adaptation rapide aux changements d'environnements
  - Coût
    - Moins onéreux qu'une supervision humaine
    - Apprentissage, ajustement automatique sans intervention humaine
  - Fonctionnalité
    - Décision prise en tenant compte de plus nombreux paramètres
    - Exploration de l'ensemble des solutions possibles plus rapidement et plus complètement

Configuration adaptative

- Configuration des consommateurs d'événements
- Définir un contrat de service pour chaque consommateur
  - Exple: 99% des événements traités en moins de 15 secondes
- Le consommateur s'adapte pour satisfaire le contrat tout en minimisant les ressources utilisées
  - Ajuster la fréquence de lecture des événements
  - Ajuster le nombre d'événements traités ensemble dans un lot
  - Ajuster le nombre de processus traitant les événements
- Le consommateur s'adapte aux variations dynamiques de l'environnement d'exécution
  - Charge (longueur des files)
  - Durée de traitement d'un événement
  - Nombre de consommateurs
    - Ajouter un consommateur → diminue la charge sur chacun

Apprentissage

- Objectif
  - améliorer le taux d'achat des utilisateurs
  - Etudier les liens navigation→ achat
- Quelle présentation des articles aboutit le plus probablement à un achat ?
  - Selon le contexte, le profil utilisateur
- Cycle d'apprentissage
  1. Etudier les achats précédents
  2. Collecter le comportement des utilisateurs
  3. Fouille de données
     - corrélations profil/présentation/achat → recommandations
  4. Répercuter les recommandations sur l'application

  Retourner à l'étape 1
- Optimisation des recommandations
  - Techniques d'optimisation issues de l'intelligence artificielle
    - Obtenir rapidement une solution quasi-optimale
  - Eviter la sur-recommandation des solutions précédemment recommandées

Stratégie 4

Prévoir les pannes

Situation de pannes

- Pannes fréquentes de tous les éléments
  - Chaque opération peut échouer
  - Chaque ressource tombera «bientôt» en panne
- Gérer les pannes le plus rapidement possible
  - Détection
  - Reprise après panne,
- Panne partielle du système
  - Ne doit pas empécher la partie sans panne de poursuivre ses traitements.

Détection des pannes

- Journal des requêtes utilisateurs
  - Le journal est généré par les applications
  - Le journal détaille les accès aux BD et aux ressource externes
  - Diffusion par message
  - Détection des pannes automatisée
    - Contrôle de l'état des applications en temps réel
      - Détection des exceptions et des alertes activées
    - Mesure de la vitesse instantanée des transactions
      - Selon l'URL, le groupe applicatif, la BD, ...

Contrôle de l'activité

Volume du journal: +1,5 tera octets / jour

Gestion des Pannes et évolution de l’application

- Généraliser le rollback
  - Toute modification doit pouvoir être défaite
- Déploiement de code
  - Bi-mensuel sur les 16 000 serveurs d’applications
  - Dépendance entre groupes applicatifs
  - Automatisation
    - Calcul des dépendances → plans de déploiement et de rollback
    - Possibilité d’abandon immédiat et automatisé
- Déploiement de fonctionnalité
  - Configuration centrale pour activer/désactiver une fonctionnalité
  - Possibilité d’activer/désactiver une fonctionnalité instantanément
  - Déploiement séparé : code / fonctionnalité
  - Impact sur les applications : tester la présence d’une fonctionnalité

Inventaire des ressources indisponibles

- Détection
  - Les applications détectent l’indisponibilité des BD et des ressources externes
  - Fausse suspicion
    - Une ressource lente mais disponible est difficile à détecter
- Opérer en mode dégradé
    - Ressource indisponibles inventoriées
      - Ne pas invoquer une ressource indisponible
      - Déclencher une alerte
    - Les fonction critiques sont re-exécutées sur une autre ressource
    - Les fonctions non critiques sont retardées
      - mise en file d’attente pour exécution ultérieure
- Inventaire explicite des ressources indisponibles
  - Reprise progressive de l’activité
  - Evite la surcharge après panne

Bibliographie

- Randy Shoup, QCon InfoQueue 2007
  http://www.infoq.com/presentations/shoup-ebay-architectural-principles



Techniques de stockage

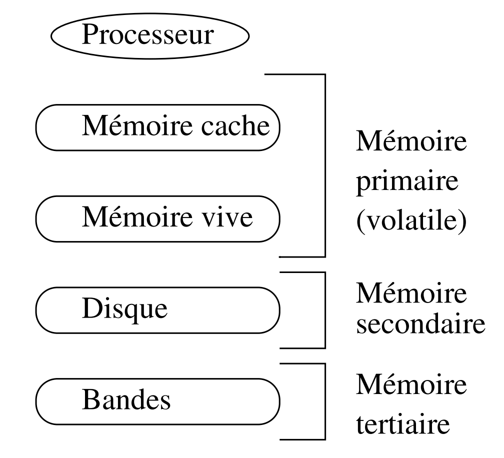
Techniques de stockage

Contenu de ce cours :

1. **Stockage de données**. Supports, fonctionnement d'un disque, technologie RAID
2. **Organisation des fichiers**. Champs, enregistrements, blocs, techniques d'accès.
3. **Un exemple concret : Oracle**.

Les mémoires d'un ordinateur

Les mémoires dans un ordinateur forment une hiérarchie :



Plus une mémoire est rapide, moins elle est volumineuse.

Quelques ordres de grandeur

Mémoire	Taille (en Mo)	Temps d'accès (secondes)
cache	Env. 1 Mo	$\approx 10^{-8}$ (10 nanosec.)
principale	$O(10^2)$ Mo	$\approx 10^{-8} - 10^{-7}$ (10-100 nanose
secondaire	$O(10^{12})$ (Gygas)	$\approx 10^{-2}$ (10 millisec.)
tertiaire	$O(10^{15})$ (Téras)	$\approx$ 1 seconde

NB : un accès disque est (environ) un million de fois plus coûteux qu'un accès en mémoire principale !

Importance pour les bases de données

- un SGBD doit ranger sur disque les données ; (parce qu’elle sont trop volumineuses ; et qu’elle doivent persister à long terme.)
- il doit les amener en mémoire pour les traiter ;
- si possible, les données utiles devraient résider le plus possible en mémoire.

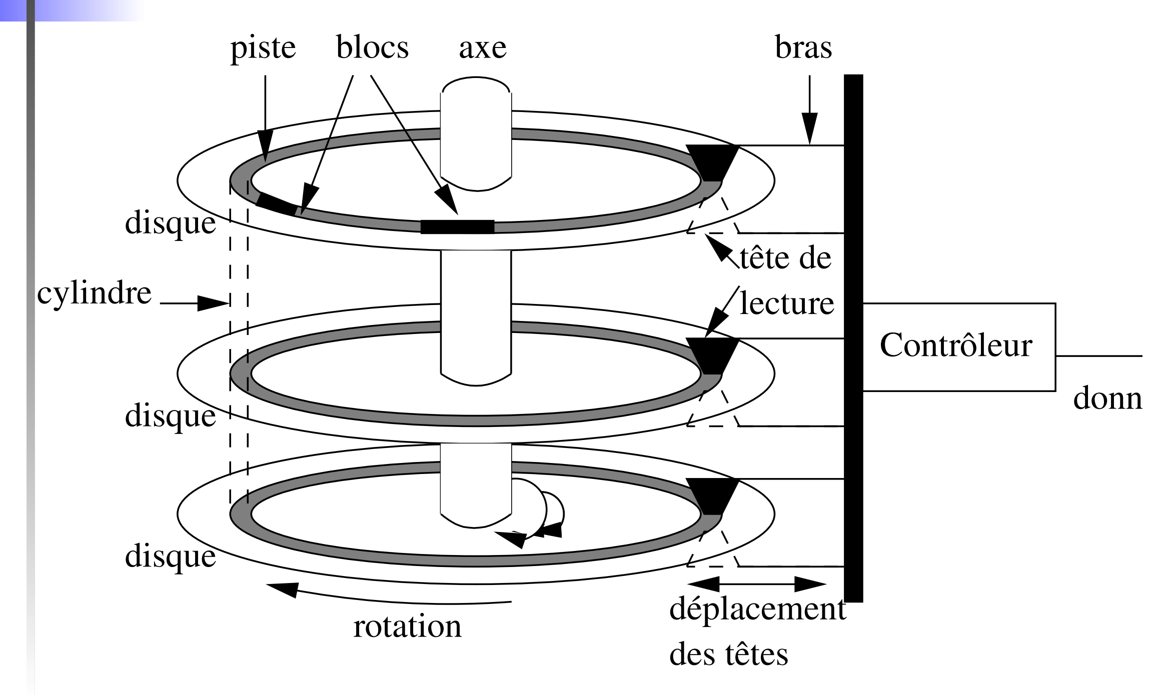
La performance d’un SGBD dépend de sa capacité à gérer efficacement les transferts disque-mémoire

Organisation d’un disque

Disque : surface magnétique, stockant des 0 ou des 1, divisé en secteurs.
Dispositif : les surfaces sont entraînées dans un mouvement de rotation ; les têtes de lecture se déplacent dans un plan fixe.

- **le bloc** est un ensemble de secteurs, sa taille est en général un multiple de 512 ;
- **la piste** est l’ensemble des blocs d’une surface lus au cours d’une rotation ;
- le **cylindre** est un ensemble de pistes situées sous les têtes de lecture.

Structure d’un disque



Disque = mémoire à accès direct

Adresse = numéro du disque ; de la piste où se trouve le bloc ; du numéro du bloc sur la piste.

1. **délai de positionnement** pour placer la tête sur la bonne piste :
2. **délai de latence** pour attendre que le bloc passe sous la tête de lecture ;
3. **temps de transfert** pour attendre que le (ou les) bloc(s) soient lus et transférés.

Important : **on lit toujours au moins un bloc, même si on ne veut qu’un octet !**

Exemple : le disque *Cheetah 18LP*, 9,1 Go

Caractéristique	Performance
Taux de transfert	80 Mo par seconde
Cache	1 Mo
Nbre de disques	3 (6 têtes)
Nombre total secteurs (512K)	17 783 438
Nombre de cylindres	9 772
Vitesse de rotation	10 000 rpm (rot. par minute)
Délai de latence	En moyenne 3 ms
Temps de positionnement moyen	5.2 ms
Déplacement de piste à piste	0.6 ms

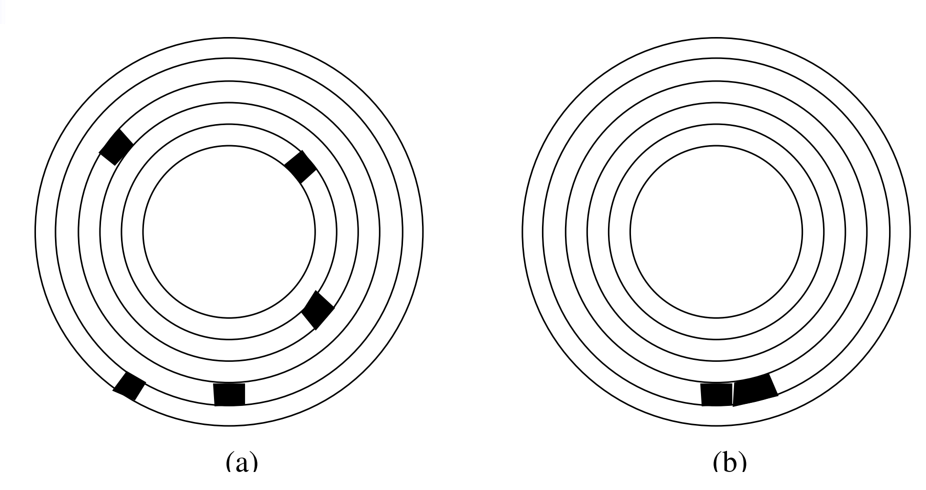
Quelques calculs...

À partir des spécifications, on calcule :

1. nombre de secteurs par face ;
2. nombre (moyen) de secteurs par piste ;
3. nombre (moyen) d'octets par piste et par cylindre ;
4. temps de rotation, et donc délai de latence ;
5. délai de transfert pour 1 ou n blocs.

On en déduit les temps *minimal*, *maximal* et *moyen* de lecture.

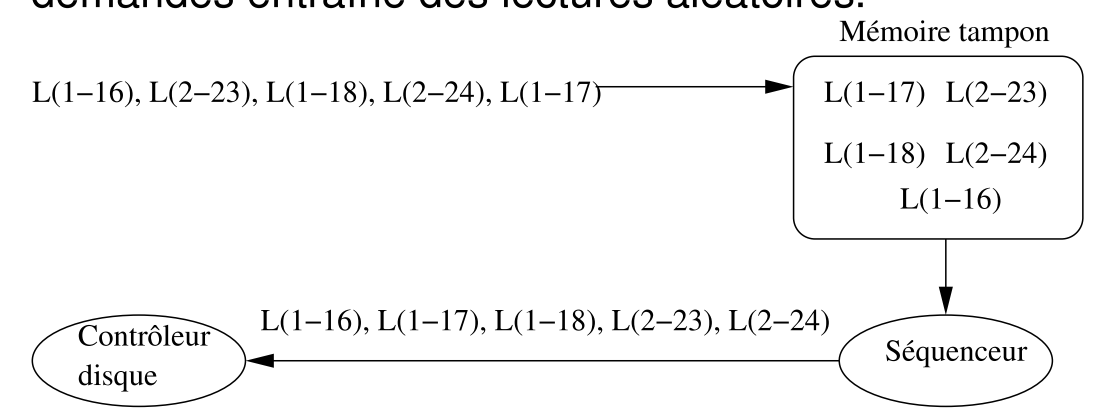
Optimisation (1) : placement des blocs



Proximité sur un disque : (1) dans le même bloc, (2) dans deux blocs consécutifs, (3) sur la même piste, (4) sur le même cylindre, (5) fonction de l'éloignement des pistes.

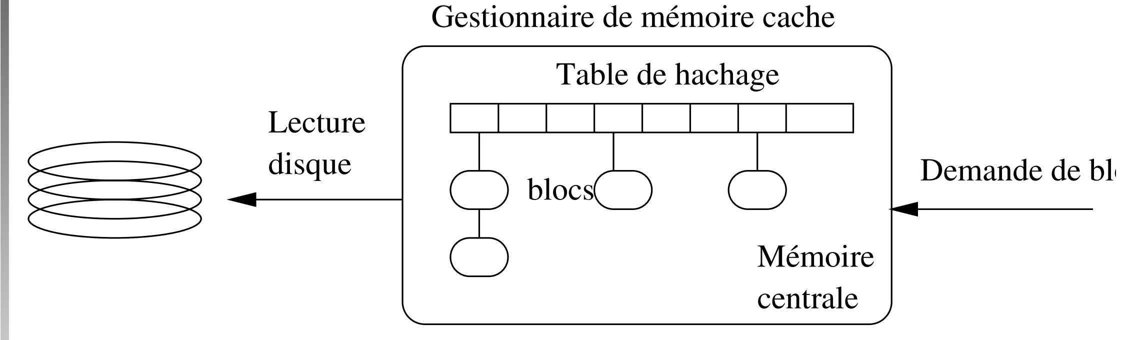
Optimisation (2) : ordre des accès

Difficulté supplémentaire : l'entrelacement des demandes entraîne des lectures aléatoires.



Optimisation (3) : mémoire cache

Dernière optimisation : garder en mémoire les blocs après utilisation.



Un des paramétrages des SGBD consiste à leur attribuer une partie de la mémoire centrale qui sert – en grande partie – de *cache*.

Technologie RAID

Stockage sur disque : point *sensible* des SGBD, pour les performances *et* pour la sécurité. On estime :

- risque de panne pour un disque pendant les prochains 10 ans = 1 ;
- risque de panne pour *deux* disques pendant les prochains 5 ans = 1 ;
- risque de panne pour *cent* disques pendant le prochain mois = 1 !

La technologie RAID vise principalement à limiter le risque dû à une défaillance.

Les niveaux RAID

Il existe 7 niveaux, numérotés de 0 à 6.

1. niveau 0 : rien !
2. niveau 1 : duplication – brutale – des données ;
3. niveau 4 : reprise sur panne basée sur la parité ;
4. niveau 5 : répartition de l'information de parité ;
5. niveau 6 : prise en compte de défaillances simultanées.

RAID 1

Principe trivial : on a deux disques, sur lesquels on écrit et lit en parallèle

- deux fois plus coûteux ;
- pas d'amélioration notable des performances.

RAID 4

Hypothèses : on dispose de n disques, tous de même structure. On introduit un *disque de contrôle* contenant *la parité*. Exemple :

D1: 11110000
D2: 10101010
D3: 00110011

Chaque bit du disque de contrôle donne la parité, pour le même bit, des autres disques :

DC: 01101001

$\Rightarrow$ permet la reprise sur panne en cas de défaillance d'un seul disque.

Performances du RAID 4

- **Lectures** : elles s'effectuent de manière standard sur les disques de données ;
- **Répartition** : le RAID 4 distribue les blocs sur les n disques, ce qui permet d'effectuer des lectures en parallèle.

Écritures il faut tenir compte des versions *avant* et *après* mise à jour d'un octet. Exemple :

avant : 11110000
après : 10011000
Octet de mise à jour : 01101000.

On doit inverser les bits 2, 3, 5 du disque de parité.

RAID 5 et RAID 6

Problème 1 du RAID 4 : n fois plus d'écritures sur le disque de contrôle.
Solution RAID 5 : les blocs de parité sont distribués sur les $n+1$ disque. Exemple :

D1: **1**1110000
D2: 1**0**101010
D3: 00**1**10011
DC: 011**0**1001

Problème 2 : et si deux disques tombent en panne en même temps ? RAID 6 : un codage plus sophistiqué permet de récupérer deux défaillances simultanées.

Fichiers

Une base de données = un ou plusieurs *fichiers*.
Un fichier = un ou plusieurs *blocs*.
Le SGBD choisit *l'organisation des fichiers* :

1. l'espace est-il bien utilisé ?
2. est-il facile et efficace de faire une *recherche* ?
3. est-il facile et efficace de faire une *mise à jour* ?
4. les données sont-elles correctement représentées, et en sécurité ?

Tous les SGBD prennent en charge la gestion des fichiers et de leur contenu.

Enregistrements

Un enregistrement = une suite de *champs* stockant les valeurs des attributs.

Type	**Taille en octets**
INTEGER	4
FLOAT	4
DOUBLE PRECISION	8
DECIMAL (M, D)	M, (D+2 si $M < D$)
CHAR(M)	M
VARCHAR(M)	L+1 avec $L \leq M$

Tailles variables et valeurs NULL

Si tous les champs sont de taille fixe et ont une valeur : pas de problème :
En pratique : certains champs ont une taille variable ou sont à NULL.

- pour les champs de taille variable : on précède la valeur par la taille exacte ;
- pour les valeurs NULL: on peut indiquer une taille 0 (Oracle) ; on peut créer un « masque » de bits.

La norme n'impose pas de choix, mais dans tous les cas il faut gérer une *information complémentaire* sur les enregistrements.

En-tête d'enregistrement

Les informations complémentaires sont stockées dans l'en-tête d'un enregistrement. Exemple :

- table *Film* (id INT, titre VARCHAR(50), année INT)
- Enregistrement (123, 'Vertigo', NULL)

Le pointeur donne par exemple l'adresse du *schéma* de l'enregistrement (de la table).

Blocs et enregistrements

- on essaie d'éviter qu'un enregistrement chevauche deux blocs ;
- on veut envisager le cas où la taille d'un enregistrement varie ;
- **on affecte une adresse à un enregistrement** pour pouvoir y accéder en une seule lecture ;
- on détermine une méthode pour gérer un déplacement.

Les deux derniers problèmes sont particulièrement cruciaux pour *l'indexation des enregistrements*.



Enregistrements de taille fixe

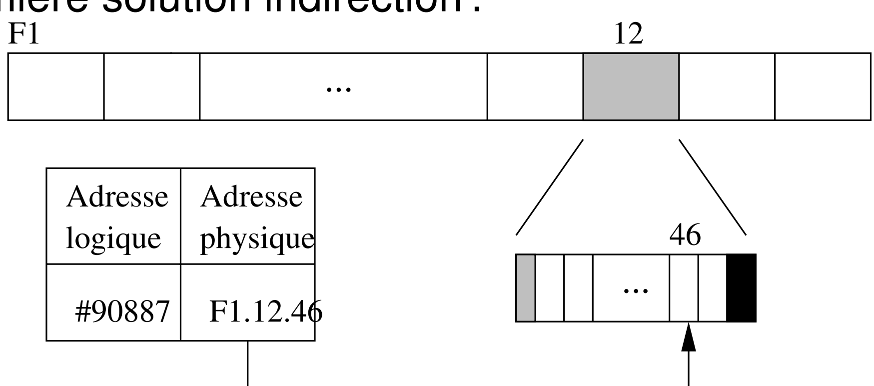
- Pour une taille de bloc B et d'enregistrement E, on a $\lfloor B/R \rfloor$ enregistrements par bloc ;
- Exemple : $B = 4096$, $E = 84$, $\lfloor \frac{4096-100}{84} \rfloor = 47$ enregistrements par bloc.
- Le 563 est dans le bloc $\lfloor 563/47 \rfloor + 1 = 12$,
- Le bloc 12 contient les enregistrements $11 \times 47 + 1 = 517$ à $12 \times 47 = 564$;
- le 563 est donc l'avant-dernier du bloc.

NB : on peut aussi le référencer par le fichier + le bloc + le numéro interne. Soit F1.12.46.

Enregistrements de taille variable

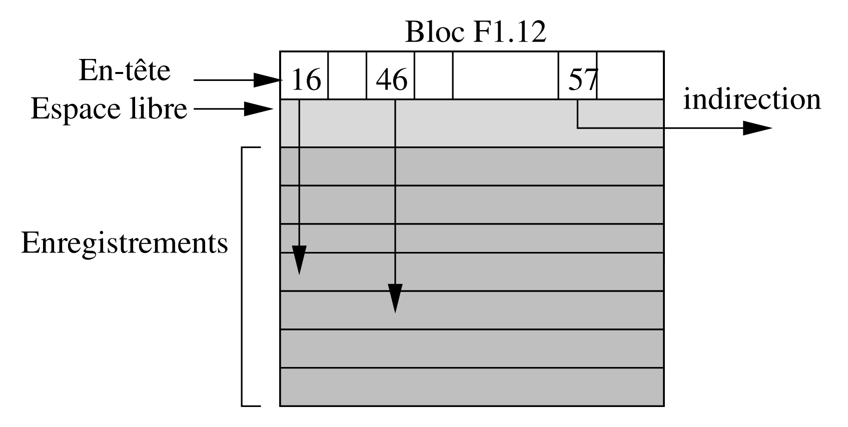
- un enregistrement peut changer de taille, avec réorganisation interne du bloc ;
- un enregistrement peut être déplacé.

Première solution indirection :



Solution intermédiaire

- on a un *adressage physique* pour le bloc ;
- au sein du bloc on a une *indirection* pour adresser les enregistrements.



Solution adoptée par Oracle.

Réorganisation du stockage

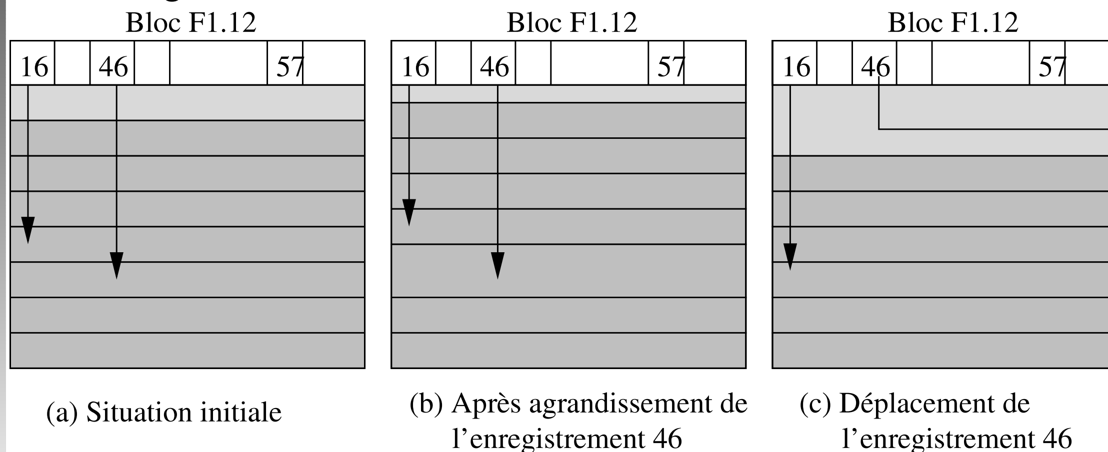
L'adressage indirect donne une certaine souplesse :

- si un enregistrement s'agrandit, mais qu'il reste de la place dans le bloc : une réorganisation interne suffit.
- sinon on le déplace *et on crée un chaînage dans l'en-tête du bloc.*

La création de chaînage pénalise les performances ⇒ si possible laisser de l'espace libre dans un bloc.

Exemple d’évolution avec chaînage

On agrandit deux fois successivement l’enregistrement F1.12.46.



(a) Situation initiale

(b) Après agrandissement de l’enregistrement 46

(c) Déplacement de l’enregistrement 46

Recherche dans un fichier

En l’absence d’index approprié, le seul moyen de rechercher un enregistrement est de parcourir séquentiellement le fichier.
La performance du parcours est conditionnée par :

- la bonne utilisation de l’espace (idéalement tous les blocs sont pleins) ;
- le stockage le plus contigu possible (même piste, même cylindre, etc).

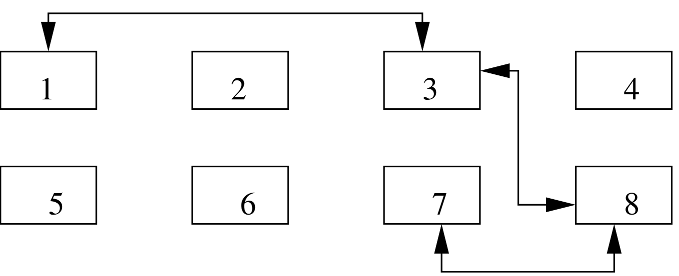
On peut faire beaucoup mieux si le fichier est trié sur la clé de recherche (recherche par dichotomie).

Mise à jour d'un fichier

Pour les UPDATE et DELETE : on se ramène à une recherche.

Pour les INSERT, problème : on ne peut pas se permettre de parcourir le fichier à chaque fois !

Première solution : liste doublement chaînée.



Mise à jour (2)

Seconde solution : garder une table des pages libres.

libre ?	espace	adresse
O	123	1
N		2
		...
O	1089	7

1

3

7

Avantage : on peut savoir facilement où trouver l'espace nécessaire.

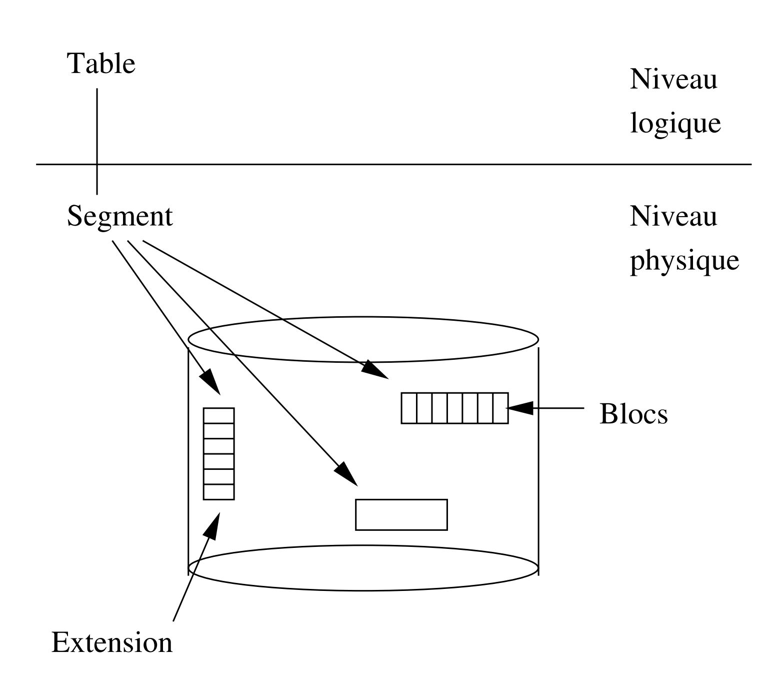
Oracle

Principales structures physiques dans ORACLE :

1. **bloc**: unité physique d'E/S.
   La taille d'un bloc ORACLE est un multiple de la taille des blocs du système sous-jacent.
2. **Extension**: ensemble de blocs contigus contenant un même type d'information.
3. **Segment**: ensemble d'extensions stockant un objet logique (une table, un index ...).

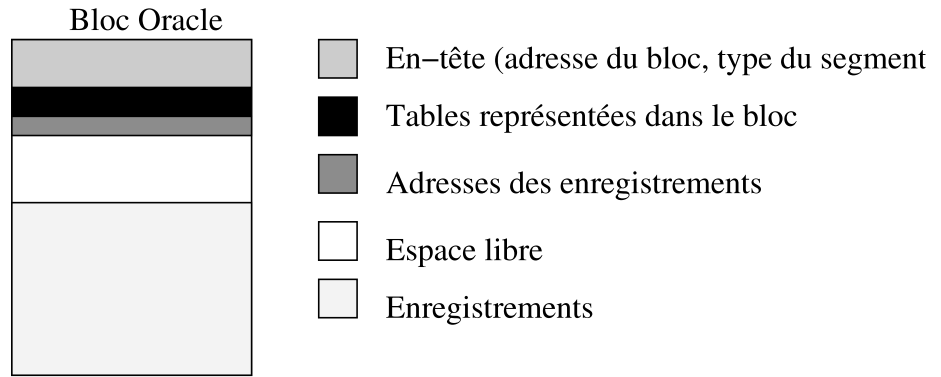
Le paramétrage du stockage des données (tables et index) est spécifié dans un *tablespace*

Tables, segments, extensions et blocs



Les blocs Oracle

La structure d'un bloc repose sur un adressage physique/logique, chaque enregistrement ayant une adresse interne.



Un chaînage est créé quand il faut déplacer un enregistrement.

Gestion de l'espace

- PCTFREE donne l'espace libre à préserver au moment de la création d'une table ou d'un index.
- PCTUSED indique à quel moment le bloc est disponible pour des insertions.

Oracle maintient un répertoire (?) des blocs disponibles pour insertions. Exemple :

1. PCTFREE = 30% et PCTUSED=70%
2. PCTFREE = 10% et PCTUSED=80%

Le second choix est plus efficace, mais plus risqué et plus coûteux.

Stockage et adressage des enregistrements

En règle générale un enregistrement est stocké dans un seul bloc.
L'adresse physique d'un enregistrement est le *ROWID* :

1. Le numéro de la page dans le **fichier**.
2. Le numéro du n-uplet dans la page.
3. Le numéro du fichier.

Exemple : 00000DD5.000.001 est l'adresse du premier n-uplet du bloc DD5 dans le premier fichier.

Extensions et segments

L'extension est une suite de blocs contigus. Le **segment** est un ensemble d'extensions contenant un objet logique.
Il existe quatre types de segments :

1. Le segment de données.
2. Le segment d'index.
3. Le *rollback segment* utilisé pour les transactions.
4. Le segment temporaire (utilisé pour les tris).

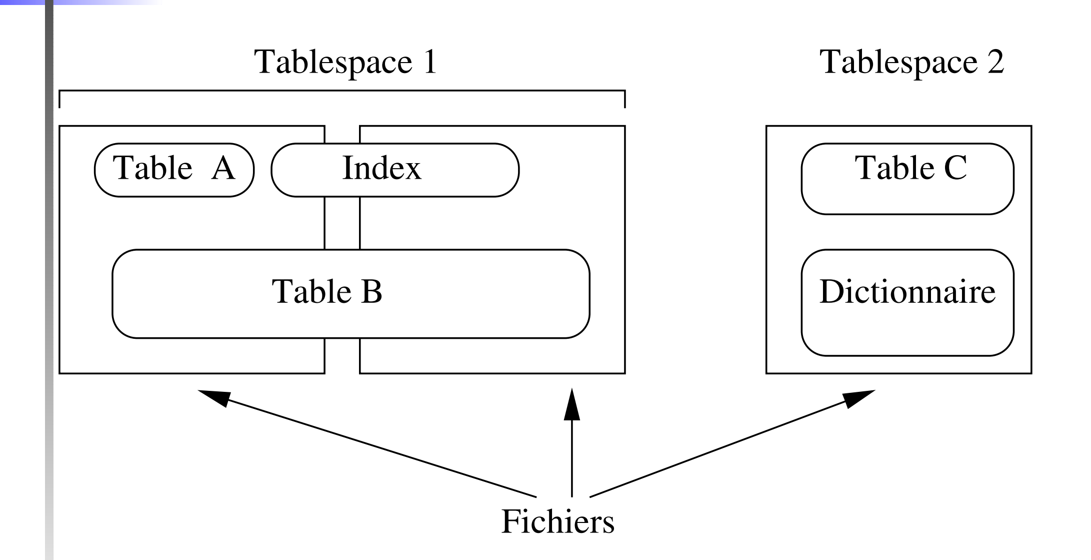
Moins il y a d'extensions dans un segment, plus il est efficace.

Les *tablespaces*

Une base est divisée par l'administrateur en *tablespace*. Chaque *tablespace* consiste en un (au moins) ou plusieurs fichiers.
La notion de *tablespace* permet :

1. De contrôler l'emplacement physique des données. (par ex. : le dictionnaire sur un disque, les données utilisateur sur un autre) ;
2. de régler l'allocation de l'espace (extensions) ;
3. de faciliter la gestion (sauvegarde, protection, etc).

Exemple de *tablespaces*



Création de *tablespaces*

```
CREATE TABLESPACE TB1
    DATAFILE 'fichierTB1.dat' SIZE 50M
    DEFAULT STORAGE (
    INITIAL 100K, NEXT 40K, MAXEXTENTS 20,
    PCTINCREASE 20);

CREATE TABLESPACE TB2
    DATAFILE 'fichierTB2.dat' SIZE 2M
    AUTOEXTEND ON NEXT 5M MAXSIZE 500M
    DEFAULT STORAGE (INITIAL 128K
        NEXT 128K MAXEXTENTS UNLIMITED);
```


Maintenance d'un *tablespace*

Quelques actions disponibles sur un *tablespace* :

1. On peut mettre un *tablespace* hors-service.
   `ALTER TABLESPACE TB1 OFFLINE;`
2. On peut mettre un *tablespace* en lecture seule.
   `ALTER TABLESPACE TB1 READ ONLY;`
3. On peut ajouter un nouveau fichier.
   `ALTER TABLESPACE ADD DATAFILE 'fichierTB1-2.dat' SIZE 300 M;`

Affectation de tables à un *tablespace*

On peut placer une table dans un *tablespace*. Elle prend alors les paramètres de stockage de ce dernier. On peut aussi remplacer certaines valeurs.

```
CREATE TABLE Film (...)
    PCTFREE 10
    PCTUSED 40
    TABLESPACE TB1
    STORAGE ( INITIAL 50K
        NEXT 50K
        MAXEXTENTS 10
        PCTINCREASE 25 );
```

Les index

ABDR, cours 3

1

Motivations

- Retrouver les données qui satisfont un critère
    - R(a,b,…),   selet * from R where b = 'Dupont'
- Accès aux données, **sans** index
    - Parcours séquentiel
        - Durée du parcours, proportionnel à la taille des données
    - Recherche par dichotomie
        - Seulement si les données sont triées (exple. tri selon b)
        - Durée : $0(\log_n)$
- Avec un index
    - Un index est une structure de données auxiliaire
    - Traverser l'index puis lire les données qui satisfont le critère
    - Un index offre un accès efficace aux données, en réduisant les lectures non nécessaires
    - Inconvénient : coût des mises à jour

2

Clé de recherche

- Critère de recherche
  - Prédicat formé de plusieurs attributs
    - Prédicat simple : attribut op valeur
      - op $\in$ {=, <, >, $\leq$, $\geq$, like}   like: compare 2 chaînes
    - Prédicat composé : and, or
- Clé de recherche
  - Les attributs du critère de recherche
  - Peut être différent de la clé de la relation
- Type de clé
  - Séquentielle : monotone selon l'ordre d'insertion
    - Ex.: date d'inscription, estampille, n° de facture
  - Non séquentielle (cas le plus fréquent)

Type de critère de recherche

- Par valeur
  - Prédicat d'égalité.   b = 'Dupont'
- Par intervalle
  - Prédicat d'inégalité.   age > 18 ,   7 < age < 77
- Par préfixe
  - like avec joker en dernière position
    - b like 'Dup%' mais pas b like 'D%t'

Type de requêtes (1)

- Requête ciblée
  - résultat = un seul tuple
  - Clé de recherche = clé de la relation
    - Critère : conjonction de prédicats d'égalité
      - Annuaire(nom, prénom, age,…), *where nom='n1' and prénom='p1'*
- Requête multipoints
  - Le résultat contient plusieurs tuples
  - Clé de recherche ≠ clé de la relation
    - Critère : conjonction de prédicats d'égalité. Ex: *where age = 23*
- Requête sur intervalle
  - Critère avec un ou plusieurs prédicats d'inégalité
    - Ex: *where age > 18*

5

Type de requêtes (2)

- Requête sur préfixe
  1) prédicat simple : *a like 'N%'*
  2) prédicat composé de *n* prédicats simples
    - Le préfixe est composé de (*n-1*) prédicats d'égalité
    - conjonction ordonnée: n-1 prédicats d'égalité suivis d'1 seul prédicat quelconque (like, <, >,...).
- Requête min-max
  - Critère: attribut = max(domaine de l'attribut)
    - Select * from Personne where age =
      - (select max(age) from Personne)
- Requête avec tri (order by)
- Requête de regroupement (group by)
- Requête avec jointure
  - Prédicat de jointure R.a=S.a

6

Structure de donnée d'un index

- Une entrée : une paire (valeur, n° de page)
- Les entrées sont organisées en arbre
- La racine de l'arbre tient en mémoire
- Les nœuds intermédiaires et les feuilles sont stockées sur le disque
  - Un index est un fichier
- Accès rapide
  - Traversée rapide depuis la racine vers les feuilles
    - Faible profondeur de l'arbre (<4)
  - Une seule lecture pour lire un nœud
    - taille d'un nœud ≤ taille d'un bloc

Structure de données (2)

- Arbre équilibré
  - B+Tree avec chaînage "horizontal" bi-directionnel des feuilles
  - Un nœud contient n valeurs croissantes et n+1 adresses des fils
  - Une feuille contient m valeurs croissantes de l'attribut
  - Pour chaque valeur m, accès :
    - soit aux adresses des tuples qui satisfont la valeur
    - soit aux tuples directement
- Tables de hachage
  - Fonction de hachage
    - Valeur v, h(v) = n° entrée dans un répertoire
    - Une entrée contient toutes les clés qui ont la même valeur h(v)
    - Pour chaque clé: liens vers les adresses des tuples
  - Statique: répertoire de taille fixe, débordement possible
  - Dynamique: le domaine de h(v) varie pour éviter les débordements
    - Ex: $h(v) = v \text{ modulo } 2^n$
      - Doubler la taille du répertoire: n → n+1
      - Diviser par 2 la taille du répertoire: n → n-1

Index / type de requête

- Quel index peut être utilisé pour quel type de requête ?
  - Requête ciblée
    - ArbreB$^+$, hachage (plus rapide que l'arbreB+ si pas de débordement)
  - Requête multi points
    - ArbreB$^+$, hachage
  - Requête sur intervalle : arbreB$^+$ grâce au chaînage des feuilles
  - Requête min max : arbreB$^+$
  - Tri sur la clé de l'index: arbreB$^+$
  - Regroupement sur la clé: arbreB$^+$, hachage
  - Equi-Jointure $R \bowtie S$ avec R.a = S.b
    - Jointure en boucle : itérer sur R, puis index sur S.b (arbreB$^+$, hachage)
    - Jointure par fusion : 2 arbres B$^+$ sur R.a et sur S.b
    - Jointure par hachage : jointure des paquets de R et S, 2 à 2
      - la fonction de hachage des index sur R.a et S.b doit être identique.

9

Inclusion : données $\subseteq$ index ?

- Traiter une requête en utilisant un index
  - Traverser l'index + lire les données
  - Racine → feuilles → adresses des tuples → tuples
- Index couvrant
  - Parcourir seulement l'index : racine → feuilles → adresses
    - sans accéder aux données : ne pas lire les tuples
  - Restreint à certaines requêtes
    - Select *projection* from ... where *prédicat*
      - Aproj = {attributs de la projection}, Apred = {attributs du prédicat}
      - La clé de l'index définit un ordre ($\prec$) sur les attributs
    - Condition pour que l'index soit couvrant
      - Aproj $\subseteq$ clé de l'index, Apred $\subseteq$ clé de l'index
      - ET les attributs de sélection précèdent ceux de projection, selon l'ordre ($\prec$) défini par l'index
      - ET toutes les valeurs des attributs doivent être dans l'index

10

Index non plaçant

- Sert à indexer un fichier non trié
  - Clé de l'index ≠ clé de tri du fichier
  - Deux tuples qui ont la même valeur de clé ne sont pas placés sur la même page
- Toujours dense
  - Une feuille pointe vers les adresses de **tous les tuples** qui satisfont la valeur
    - Feuille: liste de couples (v,p)
    - v: valeur indexée, p: page contenant la liste des adresses des tuples qui satisfont v
  - Tous les tuples sont indexés
      - A chaque tuple, correspond une adresse référencée par une valeur de clé dans une feuille.
- Plusieurs index non plaçants
    - appelés index secondaires

11

Index plaçant

- Sert à indexer un fichier trié
  - Clé de l'index ⊆ clé de tri du fichier
- Généralement non dense
  - Une feuille contient une liste de couples (v,p)
    - V: valeur indexée, p :adresse de la page contenant le premier tuple qui satisfait v
    - Ok car les tuples de la même page ont une valeur proche
  - Les valeurs du domaine de l'attribut indexé ne sont pas toutes présentes dans l'index
    - Une valeur par page suffit (max)
    - L'index contient la plus petite valeur de chaque page de données
    - Ne pas indexer 2 fois la même valeur dans 2 pages consécutives
- Peut être dense
  - si le nb de tuples ayant la même valeur de clé > nb de tuples par page
  - Pour être couvrant
- Un seul index plaçant par relation
    - appelé index primaire
    - si la relation R est répliquée : un index plaçant par réplique

12

Efficacité Index / parcours séquentiel

Relation T, nb de pages : page(T) n tuples par page
Requête Req, nb tuples du résultat : card(Req)
Index, nb de pages : P_{ind}

- Lecture séquentielle
  - Lire page(T), avec prefetch des pages de la même piste : lecture *c* fois plus rapide qu'une lecture aléatoire. (5< *c* < 10)
- Index non plaçant
  - lire 1 page par tuple,
  - Répartition aléatoire des pages lues → lecture non séquentielle
  - efficace si (card(Req) + P_{IND}) < page(T)/ c
    - P_{IND} : nombre de page de l'index qui ne sont pas déjà en mémoire
- Index plaçant
  - lire 1 page pour n tuples successifs car ils sont placés dans la même page
  - lecture séquentielle si pas de débordement
  - efficace si (card(Req)/n + P_{IND})< nb page relation

Insertion avec index

- Inconvénients
  - Une mise à jour
    - n mise à jour (des n index)
- Avantage
  - Verrouillage pour les update
    - verrou partagé sur l'index
    - verrou exclusif sur 1 seule page de données
    - moins de verrous → plus de concurrence
  - Cohérence : unicité, intégrité référentielle
    - mise à jour d'une référence avec vérification de l'existence de la clé.
    - mise à jour d'une clé avec mise à jour en cascade des références

Index multi-attributs

- Clé composée d'au moins 2 attributs
  - R(a,b, ...),   Index sur les attributs (a,b)   relation d'ordre $a \prec b$
- Structure imbriquée (hierarchie : un niveau par attribut)
  - Niveau principal : Index sur 1[er] attribut
    - Pour chaque valeur du 1[er] attribut *a*
      - Index le 2[ème] attribut *b*
      - Seuls les tuples qui ont la même valeur pour *a* sont indexés
  - Index sur chaque attribut: B+Tree ou table de hachage
- Structure en hypercube
  - Grid-file: une dimension par attribut de l'index
  - Une fonction de hachage par attribut
  - Ex. en 2 dimensions : abscisse: $x = h_1(a)$ ordonnée: $y = h_2(b)$
    - (x,y) point d'accès aux tuples
- Sert pour les requêtes avec préfixe
  - Peu d'intérêt si l'ordre des attributs de la requête diffère de l'ordre des attributs de l'index
  - a=10 and b < 3   → l'index est efficace
  - a<10 and b=3   → l'index(a,b) est inefficace
    - car cela demande de parcourir de nombreux index sur b (un pour chaque valeur de *a* inférieure à 10)

Index bitmap

- Matrice M(i,j) index sur l'attribut b de relation R(a,b,...)
  - une colonne par valeur de la clé. Nb de colonnes = nb de valeurs distinctes de la clé $\{d_1, ...d_n\}$
  - une ligne par tuple indexé. Nb de lignes = card(R)
  - un seul "1" par ligne. Soit le tuple ti $[v_a, v_b,...]$ on a M(i,j)=1 ssi $t_i.v_b = d_j$
- Efficacité
  - Prédicat simple
    - Ex: where R.b = 'Nord' où 'Nord' est la j[éme] valeur du domaine
    - Vecteur vj : j[ème] colonne de M.   Tuple ti ∈ résultat ssi vj[i]=1
  - prédicat composé
    - Where R.a=v1 and R.b=v2 → calculer l'op *et* logique entre 2 vecteurs de bits
      - Plus rapide que l'intersection entre 2 ensembles d'adresses de tuples
  - requête de dénombrement (count)
    - Pas d'accès aux données
  - Faible taille de la matrice si le domaine a peu de valeurs
    - Ex: genre {M, F}, orientation {Nord, Sud, Est, Ouest}

Implem des index dans les SGBD

- Index plaçant
  - B+Tree dense : IBM, Oracle
  - B+Tree non dense : MS, Sybase
  - Hash dense: Oracle
- Index non plaçant
  - B+Tree: IBM, Oracle,MS, Sybase
  - Bitmap, fonction : Oracle

17

Pratique

Les index avec Oracle

18

Plan d'exécution d'une requête

- Préparer l'environnement de travail
  - Créer le rôle plustrace
    - un seul rôle par base
    - en tant que sys, exécuter ...\sqlplus\admin\plustrce.sql
      - sqlplus "sys/password@base01 as sysdba" @plustrce
  - Attribuer le rôle plustrace à l'utilisateur *u1*
    - grant plustrace to u1;
  - Créer une table pour stocker les plans d'exécution
    - ouvrir une session en tant qu'utilisateur u1
    - exécuter ...\rdbms\admin\utlxplan.sql

Visualiser le plan d'exécution

- Client sqlplus : paramètre autotrace
  - plan + exécution + affiche le résultat de la requête
    - set autotrace on
  - plan + exécution de la requête mais **sans afficher** le résultat de la requête.
    - set autotrace traceonly
  - plan de la requête mais **sans l'exécuter**.
    - set autotrace on explain
  - résultat de la requête, sans autre info.
    - set autotrace off
    - nécessaire pour interroger les vues du dictionnaire
  - Compatibilité C/S
    - Utiliser le client 10G pour visualiser un plan sur le serveur 10G
    - autotrace: erreur si client 8i vers serveur 10G (sauf si rôle DBA)
- Client Jdeveloper

Trace du plan d'exécution

- Arbre d'opérations
  - ordre d'exécution: depuis les feuilles vers la racine
- Affichage tabulaire
  - une opération par ligne, lien (n° père, n° fils)
- Nom de l'algorithme
  - accès par index
    - suivi d'un accès aux données (access by rowid)
  - projection
- Estimation du coût d'une opération (coût processeur + accès disque)
  - coût relatif aux plans d'une requête
  - seulement si les statistiques ont été collectées
    - analyze table .... compute statistics
  - seulement pour l'optimisation basée sur le coût
    - 8i, 9i: alter session set optimizer_mode = choose   ou la directive /*+ choose */

21

Trace : coût réel d'exécution

- Le mode autotrace retourne la mesure de l'exécution de la requête
  - nombre de pages disque lues/écrites
    - physical read/write
    - unité de mesure: le bloc (8KO par défaut)
    - lecture d'une page disque seulement si elle n'est pas en mémoire
  - nb de pages lues en mémoire
    - logical gets
  - nb d'octets transférés : client ↔ SGBD
  - nb de verrous
  - ...

22

Trace: coût d'exécution à froid

- Traitement d'une requête
  - Lecture tuple → lecture bloc
    - accès cache si bloc en mémoire
    - Accès disque sinon
  - 1ère exécution: 100% accès disque
    - Traitement à froid
  - nème exécution: possibilité d'accès au cache
    - Traitement à chaud, plus rapide
- Mesure de performance
  - Traitement à froid, initialement: aucune donnée en mémoire
    - taille du cache : db_cache_size, v$sga
    - Quota: create profile ... private_sga
  - Vider le cache des blocs de données
    - alter system flush buffer_cache
  - Vider le cache des plans d'exécution (shared pool)
    - alter system flush buffer_pool
    - vue v$db_object_cache, package dbms_shared_pool

Directive pour l'optimiseur

- L'optimiseur détermine le plan d'exécution
  - choix d'un plan parmi n candidats
- Objectif: mesurer l'efficacité d'un plan
  - comparer l'exécution de plusieurs candidats
  - Un index est-t-il efficace pour un certain type de requête ?
    - comparer 2 plans avec et sans l'index (sans le supprimer)
- Directive : indique quel plan choisir
  - Syntaxe: select **/*+** directives ***/** ..... from
  - voir le livre
    - Performance Tuning Guide and Reference

Directives d'accès aux données

- Parcours séquentiel d'une relation
  - full(table)
- Accès par index
  - index(table nom_index)
- Ne pas utiliser l'index
  - no_index(table nom_index)
- Parcours ascendant des feuilles de l'index
  - index_asc(table nom_index)

25

26

Contrôle de concurrence

ABDR, cours 4

Concurrence

- 1 transaction: 1 séquence d'opérations
  - begin, L(A), L(B), E(B), L(C), E(A), ..., commit
- Exécution simultanée de plusieurs transactions: T_1, ..., T_n
  - A un instant t : plusieurs transactions ont commencé mais ne sont pas encore validées

Exemple

- 2 transactions
  - T1: L(A), A←A+100, E(A), L(B), B←B+100, E(B)
  - T2: L(A), A←2A, E(A), L(B), B←2B, E(B)
  - Contrainte de cohérence: A=B
- BD: état initial A=25 B=25
  a) T1 puis T2 : A=250, B=250
  b) T2 puis T1 : A=150, B=150
  c) L_1(A), A←A+100, E_1(A), L_2(A), A←2A, E_2(A), L_1(B), B←B+100, E_1(B), L_2(B), B←2B, E_2(B) : A=250, B=250
  d) L_1(A), A←A+100, E_1(A), L_2(A), A←2A, E_2(A), L_2(B), B←2B, E_2(B), L_1(B), B←B+100, E_1(B) : A=250, B=150

Besoins

- Eviter les incohérences
  - Une transaction ne doit pas **modifier** les données qui sont en cours de modif par une autre trans.
    - E_1(A), E_2(A), valid(T_1), valid(T_2) : Écrasement: perte d'une écriture
  - T_1 ne doit pas **lire** les données en cours de modif par T_2.
    - E_2(A), L_1(A), abandon(T_2), valid(T_1)
      - T_1 a lu une donnée qui n'a jamais été validée. Lecture sale.
  - T_1 peut-elle lire des données modifiées récemment ?
    - L_1(A), E_2(A), valid(T_2), L_1(A)
      - T_1 lit 2 valeurs différentes de A. Lecture non répétable
      - Récemment: après le début de T_1, après la première L_1(A)
    - L(A) = {Etudiant tq. age<23}. L_1(A), insert_2([Dupond, 21]), L_1(A)
      - T_1 'voit' un nouveau tuple. Lecture fantôme.

Définitions

- Opération
  - $L_i(X)$, $E_j(Y)$
  - conflit ssi : même donnée, 2 trans différentes et au moins 1 écriture.
    - L_1A confl E_2A E_1A confl L_2A E_1A confl E_2A
    - 2 opérations en conflit ne sont pas permutables
- Séquence
  - Suite ordonnée d'opérations appartenant à plusieurs transactions
  - Séquence en série (sérielle)
    - opérations regroupées par transaction
    - jamais une L/E_i entre deux L/E_j
  - Séquence sérialisable (sériable)
    - équivalent à une séquence sérielle obtenue par des permutations d'opérations non conflictuelles

5

Précédence

- Graphe de précédence
  - nœud: transaction
  - arc: Ti→Tj
    - L/Ei est en conflit avec L/Ej   et   L/Ei précède L/Ej
- Théorème

  Graphe de précédence acyclique

  ⇔ séquence sérialisable

6

Exécution des transactions

- Garantir une exécution sérialisable
  - vérifier que le graphe de précédence est acyclique
    - pour les traitements batch, exécution ultérieure
  - empêcher les cycles
    - traitements en ligne, exécution immédiate

Verrouillage

- Verrous
  - obtenir un V exclusif (Vx) avant une E
  - obtenir un V partagé (Vp) avant une L
- 2PL: verrouillage en 2 phases
  - phase 1: acquisition des verrous
  - phase 2: relâchement des verrous

  ⇒ Jamais d'acquisition après un relâchement
- Une séquence obtenue avec verrouillage 2PL est sérialisable
- Inconvénient
  - interblocage
    - $L_1(A)$, A←A+100, $E_1(A)$, $L_2(B)$, B←2B, $E_2(B)$, ... $L_1(B)$?
    - Résolution: abandonner une transaction

Objectifs du contrôle de concurrence

- Cohérence des données
  - sérialisabilité: chaque transaction est isolée des autres
    - $\forall$ T , T ne peut pas accéder aux mises à jour des autres trans.
- Performance
  - réduire le blocage
    - attente d'un verrou détenu par une autre transaction
  - éviter les interblocages
- Compromis
  - moins d'isolation, moins de blocage, meilleure perf

9

Niveau d'isolation

- la norme SQL92 définit 4 niveaux

Isolation Level	**Dirty Read**	**Non repeatable read**	**Phantom Read**
read uncommitted	oui	oui	oui
read committed	non	oui	oui
repeatable	non	non	oui
serializable	non	non	non

10

Isolation avec verrouillage

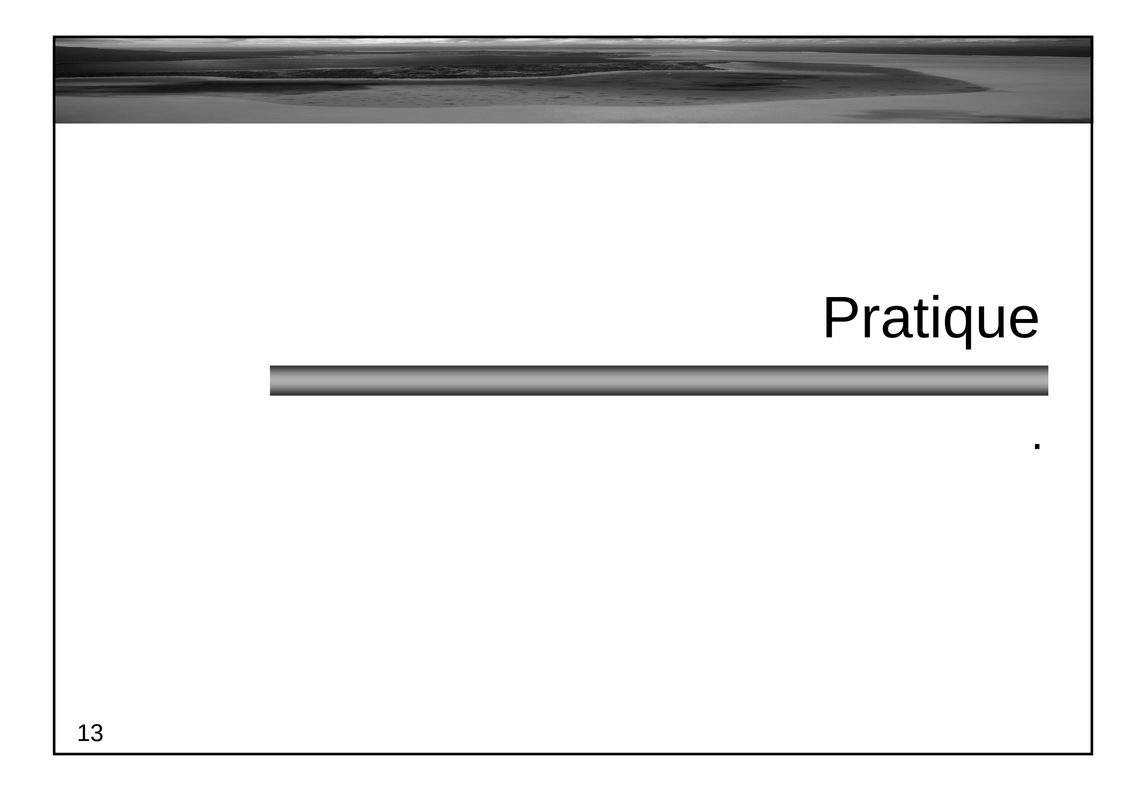
- Read commited
  - Vp avant lecture, Vx avant écriture
  - relâcher Vp après une lecture, Vx en fin de transaction
- Serializable
  - Vp avant lecture, Vx avant écriture
  - 2PL : relâcher Vp et Vx en fin de transaction
  - Granularité des verrous
    - Verrou sur une relation
    - Verrous sur un tuple + verrous sur les index
- Performance
  - Problème 1: les lectures peuvent bloquer les écritures
  - Solution: ne pas accorder de Vp en cas d'attente de Vx
    - Tester si attente Vx > durée max
  - Problème 2: Abandon après relâchement des verrous
    - Abandon en cascade des transactions qui ont effectué une lecture sale
  - Solution : verrouillage strict = ne pas relâcher les verrous avant le commit

11

Isolation avec gestion de version

- Objectif
  - garantir un niveau d'isolation sans prendre de verrou en lecture
  - aucun Vp : lecture non bloquante et non bloquée
  - Vx avant écriture, relâché en fin de transaction
  - maintenir plusieurs versions des données
    - Version d'un tuple: (tuple, transaction) avec transaction(date début, validée)
- Niveau read commited
  - Lire la dernière version validée
- Niveau serializable
  - Lecture: lire la dernière version validée avant la date de début de la transaction
  - Ecriture: refus si la date de la dernière version validée est plus récente que la date de début de la transaction
- Implémentation
  - Oracle: snapshot isolation
  - Date de début = date de la première instruction sql de la transaction
  - Dernière version validée toujours disponible dans l'espace *undo tablespace*
  - Reconstruire une version antérieure à la dernière version validée à partir du journal (undo/redo log)

12



Niveau d'isolation

- client sqlplus
  - paramètre isolation_level
  - alter session set isolation_level = serializable;
  - ....   read committed;
- client jdbc
  - java.sql.Connection
    - setAutoCommit(false);
    - setTransactionIsolation(TRANSACTION_SERIALIZABLE);
    - commit() ou rollback()

14

Verrous

- R(a, b)
- Obtenir un Vx sur un tuple de R
  - update R set ... where a= ... ;
  - select * from R where a = ... for update;
  - PL/SQL : package dbms_lock
- Obtenir un Vx sur une table entière
  - Lock table
- Relâcher les verrous
  - Relâchement en **fin** de transaction
    - commit;
    - marque le début de la transaction suivante
  - Relâchement **avant** la fin de la transaction
    - Rollback to savepoint ... ;
    - PL/SQL : dbms_lock.release(...)

Performance des requêtes

ABDR, cours 5

Référence

- D. Shasha, P. Bonnet
  - Chapitre: Query Tuning, 2001

Motivations

- Requête complexe
  - regroupement, tri
  - agrégation (min, max, avg), fonction analytique
  - sous requêtes imbriquées
    - select x, (select y from ....), z
    - from (select ...) x,
    - where x.a in (select ...), where exists (select ...)
  - multi relations
  - Prédicat complexe
    - composé : and, or, between
    - fonctions, expr math

Exemple

```
■ SELECT  s.RESTAURANT_NAME, t.TABLE_SEATING, to_char(t.DATE_TIME,'Dy, Mon FMDD') AS THEDATE,
  to_char(t.DATE_TIME,'HH:MI PM') AS THETIME,to_char(t.DISCOUNT,'99') || '%' AS AMOUNTVALUE,t.TABLE_ID,
  s.SUPPLIER_ID, t.DATE_TIME, to_number(to_char(t.DATE_TIME,'SSSSS')) AS SORTTIME
■ FROM TABLES_AVAILABLE t, SUPPLIER_INFO s,
    ■          (SELECT      s.SUPPLIER_ID, t.TABLE_SEATING, t.DATE_TIME, max(t.DISCOUNT) AMOUNT, t.OFFER_TYPE
                FROM   TABLES_AVAILABLE t, SUPPLIER_INFO
                WHERE       t.SUPPLIER_ID = s.SUPPLIER_ID
                   and (TO_CHAR(t.DATE_TIME, 'MM/DD/YYYY') !=
          TO_CHAR(sysdate, 'MM/DD/YYYY') OR TO_NUMBER(TO_CHAR(sysdate, 'SSSSS')) < s.NOTIFICATION_TIME - s.TZ_OFFSET)
                   and t.NUM_OFFERS  > 0
                   and t.DATE_TIME > SYSDATE
                   and s.CITY = 'SF'
                   and t.TABLE_SEATING = '2'
                   and t.DATE_TIME between sysdate and (sysdate + 7)
                   and to_number(to_char(t.DATE_TIME, 'SSSSS')) between 39600 and 82800
                   and t.OFFER_TYPE = 'Discount'
               GROUP BY
                 s.SUPPLIER_ID, t.TABLE_SEATING, t.DATE_TIME, t.OFFER_TYP
              ) u
■ WHERE
            t.SUPPLIER_ID   = s.SUPPLIER_ID
           and u.SUPPLIER_ID   = s.SUPPLIER_ID
           and t.SUPPLIER_ID   = u.SUPPLIER_ID
           and t.TABLE_SEATING = u.TABLE_SEATING
           and t.DATE_TIME     = u.DATE_TIME
           and t.DISCOUNT = u.AMOUNT
           and t.OFFER_TYPE    = u.OFFER_TYPE
           and (TO_CHAR(t.DATE_TIME, 'MM/DD/YYYY') != TO_CHAR(sysdate, 'MM/DD/YYYY') OR
          TO_NUMBER(TO_CHAR(sysdate, 'SSSSS'))  < s.NOTIFICATION_TIME - s.TZ_OFFSET)
           and t.NUM_OFFERS    >
           and t.DATE_TIME > SYSDATE and s.CITY = 'SF' and t.TABLE_SEATING = '2'
           and t.DATE_TIME between sysdate and (sysdate + 7)
           and to_number(to_char(t.DATE_TIME, 'SSSSS')) between 39600 and 82800 and t.OFFER_TYPE = 'Discount'
■ ORDER  BY AMOUNTVALUE DESC,  t.TABLE_SEATING ASC, upper(s.RESTAURANT_NAME) ASC,SORTTIME ASC,
  t.DATE_TIME ASC
```

Traitement de requête

- Requête → plan d'exécution
  - Arbre d'opérateurs
  - Blocs de requêtes simples : une seule clause select
  - Ordre d'exécution
  - Algorithme d'un opérateur
- Etapes du traitement
  - Requête → blocs simples
  - Bloc → arbre d'opérateurs
  - Opérateur → algorithme

5

Méthode d'accès aux données

- Parcours séquentiel
  - Possible pour toute relation
- Usage d'un index
  - si la requête contient la clé de l'index
- Choix parmi plusieurs méthodes d'accès

6

Ordre de traitement des données

- Opérateur unaire
  - sélection σ_{pred} projection: π_{Attr}
  - distinct δ_{Attr}
  - Ordre équivalent
    - Composition commutative: $\sigma_p (\pi_A(R)) = \pi_A (\sigma_p (R))$ si $\{attr(p)\} \subseteq A$
    - Simplification: $\pi_A (\pi_B(R)) = \pi_A (R)$
- Opérateur n-aire
  - Jointure, union
  - Ordre équivalent
    - Associativité, commutativité
    - Distributivité union ./. jointure
- Choix d'un ordre
  - Optimisation de requête

7

Optimisation de requêtes

- Objectif
  - Théorique: Déterminer le plan optimal
  - Pratique: Eviter les plans trop lents
- Principe
  - Enumérer les plans équivalents
    - Tous les plans: exhaustif
      - n! ordres équivalents pour joindre n relations
    - 1 seul plan
      - Ordre jointure = celui de la clause from
      - sélection et projection avant jointure
      - sélection avant projection si index
    - Certains plans
      - Enumérer des sous plans : hyp: sous plans optimaux ⇒ plan optimal
  - Choisir un plan
    - Modèle de coût

8

Visualiser l'exécution d'une requête

- Visualiser le plan d'exécution
  - Ordre des opérateurs
  - Méthode d'accès
  - Algorithmes
  - plan choisi par l'optimiseur de requêtes ≠ plan prévu ?
- Détecter une requête lente
  - Accès inefficace
    - Index non utilisé
  - Perf
    - nb de pages de données lues trop important
    - Nb d'octets transféré trop important
    - Consomme trop de ressource: CPU, disque

Modifier l'exécution d'une requête

- Objectif
  - Forcer l'optimiseur à construire un certain plan d'exécution
- Type d'action
  - Modification locale : sans effet de bord
  - Modification globale : effet global
    - Effet nuisible sur les autres requêtes, transactions, applications
    - Compromis: comparer gain local vs perte globale
- Action locale : modifier la requête SQL
  - Factorisation, désimbrication
    - with *nom_requête_factorisée* as (select ...) select ...
  - Directive d'optimisation : /*+ directive */
- Action globale
  - modifier les méthodes d'accès
    - Ajouter/supprimer un index
    - Reconstruire un index pour le défragmenter
    - Changer la structure arbreB+ ⇄ hachage
  - modifier le schéma

Exemple

- Schéma
  - Employé(num, nom, dir, serv, salaire, contact)
      - *contact* : le nombre d'amis
    - Index
      - Index plaçant sur num (clé d'*Employé*)
      - 2 index secondaires I1sur nom, I2: serv
  - Etu(num, nom, dipl, année)
    - Index
      - Index plaçant sur num (clé d'*Etu*)
      - Index secondaire sur nom
  - Service(ns, dir, addr)
      - Index plaçant sur ns (clé de *Service*)

Réécriture de requêtes

- Réécriture manuelle
  - En amont de la réécriture effectuée par l'optimiseur
  - Palier les manques de l'optimiseur
  - Orienter la réécriture effectuée par l'optimiseur
- Cas de réécriture
  - Forcer l'usage d'un index
  - Eliminer les tris inutiles (distincts, order by)
  - Réduire le nb de sous-requêtes (corrélées ou non)
  - Utiliser explicitement une relation temporaire
  - Conditions de jointure
  - Eliminer une clause *having*
  - Réécriture en présence de vues
  - Redondance : ajouter des vues matérialisés

Usage d'un index

- L'optimiseur ne choisit pas d'index si
  - Expression arithmétique
    - Ex: salaire mensuel: where sal/12 > 1000
      - ⇔ where sal > 12*1000
  - Fonctions sur des chaînes
    - Ex: where substr(name,1,1) = 'G'
      - ⇔ where name like 'G%'
  - Prédicat de comparaison
    - types numériques différents
    - Comparaison avec la valeur NULL

13

Eliminer les distincts inutiles

- Requête
  - Le n° des employés travaillant dans un des services de la relation Service ?
    - Select **distinct** num
    - from Emp e, Serv s
    - Where e.serv = s.ns
  - Le mot *distinct* n'est pas nécessaire
    - Contrainte : e.serv fait référence à s.ns
      - Un emp travaille dans un seul service
      - Jointure naturelle : num est une clé de E ⋈ S
      - Projection sur clé : aucun double
    - Sans contrainte de référence
      - un employé peut travailler dans un service non référencé
      - card(E ⋈ S) < card(E) , le distinct demeure inutile

14

Ensemble unique en SQL

- Une requête SQL contient-elle des doubles ?
  - Req SQL ⇔ ensemble unique ?
- R: Select A from T where ...
  - Clé(T) ⊆ A ⇒ R est unique
  - Généralisation: Clé(T) ⊆ A ⇒ T est privilégiée
- Atteinte
  - Si la jointure de S avec T ne modifie pas l'unicité de T alors S atteint T
    - S $\bowtie_A$ T clé(S) ⊆ A ⇒ S→T
  - Transitivité: (S→T et T→U) ⇒ S→U
  - Atteinte non commutative
- Théorème
  - Requête: select A from R1,...,Rn where... Requête unique ssi
    - ∀Ri, Clé(Ri) ⊆ A
    - OU ∀Rj, Clé(Rj) ∩ A ≠ Ø, ∃Ri, Clé(Ri) ⊆ A ∧ Ri → Rj

15

Exemple 1

- Requête
  - le n° des Emp dirigés par un directeur de service
    - Select num
    - From Emp e, Serv s
    - Where e.dir = s.dir
- Unicité ?
  - Emp est priviliégiée
  - Serv n'est pas privilégiée et n'atteint pas Emp
    - Plusieurs services pour un directeur
  - Pas d'unicité car
    - Un tuple d'employé est joint avec tous les services dirigés par le directeur de l'employé

16

Exemple 2

- Requête
  - Select e.num, s.ns
  - From Emp e, Serv s
  - Where e.dir = s.dir
- Unicité
  - Oui car E et S sont privilégiées
  - Unicité sur les couples (num,ns) mais pas sur num seul

Exemple 3

- Requête
  - Select t.num
  - From Etu t, Emp e, Serv s
  - Where t.nom = e.nom and e.serv = s.ns
- Unicité ?
  - Etu est privilégiée, Serv atteint Emp
  - mais Emp n'atteint pas Etu
    - car e.nom n'est pas une clé d'Employé
  - Non unique: le *distinct* est nécessaire pour assurer l'unicité

Requête imbriquée

- Imbrication
  - Une requête englobante contient une sous requête
  - Une relation est dite
    - Englobante si elle ∈ clause *from* de la requête englobante
    - Imbriquée " " " imbriquée
- Sous requête corrélée
    - La sous requête fait référence à une variable définie dans la requête englobante.
- Type de requêtes imbriquées
  - Sous requête corrélée
    - Sous requête Avec / Sans agrégat
  - Sous requête non corrélée (NC)
    - Sous requête Avec / Sans agrégat

Sous requête NC sans agrégat

- Réécriture
  - Fusionner les 2 clauses from
  - Compléter la clause where
    - Prédicat composé: conjonction ajouter un prédicat de jointure
  - Vérifier que la cardinalité est conservée
    - Conservation si chaque relation imbriquée atteint une relation englobante
      - ∀Ri imbriquée, ∃Rj englobante, Ri → Rj
    - Si requête englobante avec agrégation
      - Alors vérifier la cardinalité avant agrégation
- Exemple
  - Requête imbriquée

    Select ... from R

    Where ... and R.a in (select S.b from S)
  - Désimbrication

    Select ... From R**, S**

    Where ... and R.a = S.b

Exemple

- R1: salaire moyen des directeurs de Service
  Select avg(sal)
  From Employé
  Where dir in (select dir from Service)
- R2 après réécriture
  Select avg(sal) from Employé e, Service s
  Where e.dir = s.dir
- R2 ≠ R1
- R3 = R1
  - Insert into T1 (select **distinct** dir from Service)
  - R3:
    select avg(sal) from Employé e, **T1** t
    Where e.dir = t.dir

Sous requête corrélée

- Requête
  - Le n° des employés qui ont un salaire égal au salaire moyen de leur service.

```
Select num
From Employé e1
Where sal = (select avg(sal)
             from Employé e2
             where e2.serv  = e1.serv
             )
```

Sous requête corrélée

- Réécriture
  - Relation temporaire T1 (sal_moyen, serv)
    - Insert into T1
    - Select avg(salary) as sal_moyen, e.serv
    - From Employé e
    - Group by e.serv
  - Puis
    - Select num
    - From Employe e, T1 t
    - Where e.serv = t.serv
    - And e.sal = t.sal_moyen

Sous requête corrélée : Limitations

- Requête R1
  - Le n°des employés dont le nb d'amis est = au nb d'employés de leur service.
  - Select num
  - From Employé e1
  - Where amis = count(select e2.num
    from Employe e2, Service s
    where e2.serv = s.ns
    and e2.serv = e1.serv
)

Sous requête corrélée : Limitations

- Réecriture non équivalente (R2 $\neq$ R1)
  - Relation temporaire T1(nbCol, serv)
    - Insert into T1 (select count(num) as nbCol, serv
      from Employé e, Serv s
      where e.serv = s.ns
      group by e.serv
    - Select num
    From Employé z, T1 t
    Where e.amis = t.nbCol
    And e.serv = t.serv
    - R2 $\subseteq$ R1 . R2 ne contient pas les Employé sans amis et qui travaillent dans un service non référencé.

25

Relation temporaire : limitations

- Requête
  - Le n° des employés du service compta gagnant plus de 1000 EUR
    - Insert into T1
    Select num, serv From Employé where sal > 1000
    - Select num
    From T1 where t1.serv = 'Compta'
  - Ordre Inefficace
    - Sélection sal: lecture séquentielle, pas d'index sur sal
    - Sélection serv: lecture séquentielle
  - Sans relation temporaire
    - Accès par index sur e.serv puis sélection sal
    - Plus efficace

26

Jointure

- Schéma
  - R(a,b,...) S(c,d,...) Index plaçant sur R.a
- Prédicat de jointure
  - R.a = S.c
  - Index plaçant sur S.c
    - Jointure par fusion directe, sans tri car données déjà triées
  - Index secondaire sur S.c
    - tri de S selon c puis fusion
    - Ou jointure par boucles imbriquées
      - itération sur R imbriquant l'accès par index S.c
- Prédicat composé
  - R.a = S.c AND S.d = v
  - prédicat S.d = v peu sélectif (80%)
    - index plaçant sur S.d, index secondaire sur S.c
      - Jointure par tri de S selon c puis fusion puis sélection sans index
      - Ou accès par index S.d puis jointure

Clause having : usage

- Clause Where vs. Having
  - Where: concerne tous les attributs des relations de la clause from
    - Ordre : where est traité **avant** le regroupement
  - Having: concerne uniquement les attributs de regroupement.
    - Ordre: having est traité **après** le regroupement
- Traiter une sélection le plus tôt possible
  - Ne pas utiliser de clause *having* si le *where* est suffisant
  - R1 : Select avg(sal) from Employé
    group by serv having serv = 'Compta'
  - R2 : Select avg(sal) from Employé
    where serv= 'Compta' group by serv
- Clause *having* nécessaire
  - clause avec calcul d'agrégat
    - Ex: Having count(num) > 100

Usage des vues

- Vue = expression SQL nommée
  - Une vue n'est pas une relation temporaire
  - Une vue n'est pas une vue matérialisée
- Requête exprimée sur des vues
  - Substitution → requête imbriquée
  - Réécriture

Requête répartie

- Requête manipulant des données distantes
  - Clause from R1, ..., Rn
    - Ri : relation distante accessible à travers le réseau
      - Lien: database link
  - Requête distante
    - Toutes les Ri sont sur la même instance distante
  - Requête répartie
    - Au moins 2 relations situées sur des instances différentes
- Vue répartie
  - requête répartie nommée
  - Généralisation d'une requête répartie
    - clause from : Ri peut être une vue répartie

Requête répartie: traitement

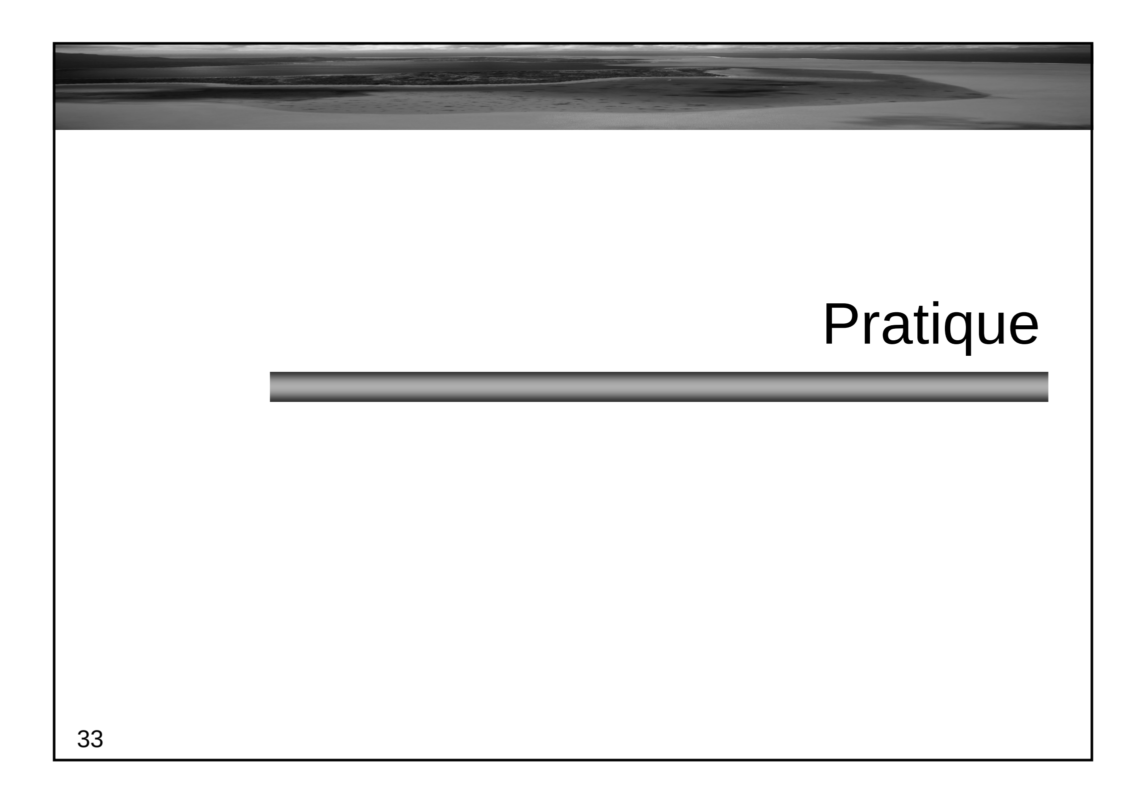
- Réécriture: requête répartie →
  - Sous requête distante, monosite.
  - Opérateur global
    - Opérande : sous requête distante
    - Choix de l'instance traitant l'opérateur global
- Ordre
  - Requête distante puis opérateur global
  - Opérateur global
    - Binaire
      - Jointure par fusion si opérandes triées
      - Jointure par boucle imbriquée
    - N-aire
      - Union

31

Traitement réparti : limitations

- Jointure globale $R \bowtie_a S$
  - jointure par boucle avec index S.a
- Simplification du plan d'exécution
  - Elagage si fragmentation des données connue
    - $V = R1 \cup R2 \Leftarrow R1 = \sigma_p (V), R2 = \sigma_{\neg p} (V)$
    - $V = R1 \bowtie R2 \Leftarrow R1 = \pi_A(V), R2 = \pi_B(V)$
  - Sinon pas d'élagage
- Transfert intersite
  - réécriture S2→S1 et S3→S1 ⇔ S2→S3→S1
    - $R \bowtie_A S = (R \ltimes_A S) \bowtie_A S = (R \bowtie_A \pi_A(S)) \bowtie_A S$
- Choix entre répliques
- Implémentation
  - Réécriture manuelle avec vues
  - Trigger, procédure stockée, appli middleware

32



Traitement réparti

- BD répartie en 3 sites
  - Site$_i$: données de l'utilisateur etunn_i connect ... show user
    - Lien intersite : create database link user_db_links
  - Sources
    - SiteB : EmpB: Employé age<20
      - Create table Emp2 as select ....
    - SiteC: EmpC: Employé age $\geq$ 20, ServiceC(ns, ...)
  - Base globale
    - SiteA: seulement des vues, aucune donnée
      - Create synonym EmpB for EmpB@siteB
      - Create view VueEmp user_views
- requête globale
  - Nombre d'employés prénommés p1
    - Select ... From VueEmp
  - Plan: identifier les transferts et le site de chaque opérateur

34

Fonctionnalités à expérimenter

- Sélection
  - Réécriture : sélection globale → sélection distante
    - Sélection avant/après transfert
  - Avec/Sans index
  - Elagage: select * from VueEmp where age > 50
- Projection :
  - Projection avant/après transfert
- Mise à jour
  - Pb mise à jour dans une vue
  - Insertion : create trigger *instead of* insert
  - Suppression
  - Update age d'un Emp 15ans → 30ans Update plusieurs Emp
- Jointure
  - EmpB $\bowtie_{serv=ns}$ ServiceC
  - Algorithme et index utilisés ?
    - Facteurs qui influencent le choix de l'algorithme ?